Alla scoperta dello ZX Spectrum



EDIZIONE ITALIANA

GRUPPO
EDITORIALE
JACKSON

a cura di
Rita Bonelli

Alla scoperta dello ZX Spectrum

a cura di
Rita Bonelli

Traduzione di:
Giacomo Bortone
e
Andrea Mazzini

Prima Edizione:
1983

**GRUPPO
EDITORIALE
JACKSON
Via Rosellini, 12
20124 Milano**

Stampato in Italia da:
S.p.A. Alberto Matarelli - Milano - Stabilimento Grafico

SOMMARIO

PREFAZIONE

Nel gennaio 1982 è stata pubblicata da questo stesso Gruppo Editoriale la "Guida al Sinclair ZX81", ora, a poco più di un anno di distanza, sto ultimando la preparazione di questo manuale dedicato al nuovo nato della famiglia Sinclair: lo ZX SPECTRUM.

Si tratta di un prodotto molto importante; abbiamo un piccolo calcolatore a colori, che risulta, in sostanza, piccolo solo nelle dimensioni.

Cominciamo a vedere quali sono i contenuti di questo libro. Sfogliando l'indice si vede che a questa prefazione seguono 35 capitoli che sono la traduzione dei 2 manuali originali in inglese dello ZX Spectrum. Di questi i primi 9 sono una introduzione al calcolatore e dovrebbero essere letti da tutti per cominciare a farsi una prima idea dello Spectrum. I principianti dovranno studiare i primi 9 capitoli con attenzione, gli altri potranno scorrerli rapidamente per cominciare a capire le differenze tra lo Spectrum e gli altri calcolatori. I capitoli da 10 a 35 contengono le notizie necessarie per imparare ad usare lo ZX Spectrum sfruttandone tutte le possibilità. Non è comunque necessario occuparsi subito di tutto, ma si può procedere per gradi, in modo di cominciare ad ottenere delle soddisfazioni dal calcolatore.

La prima cosa da fare è quella di imparare a scrivere un programma in Basic. Per raggiungere questo scopo (ovviamente questi discorsi valgono solo per i principianti) possono essere inizialmente trascurati alcuni capitoli, come il 19 sulle funzioni matematiche, il 21 sulle variabili con indice (almeno nella prima fase dello studio), il 23 sui caratteri e su alcune funzioni avanzate del linguaggio, il 24 sulle finezze della PRINT e della INPUT, il 25 sugli ATTRIBUTI delle posizioni del video, il 26 sulle possibilità grafiche, il 27 sul movimento, e quelli dal 31 al 35. Si raccomanda, invece di studiare almeno la prima parte del capitolo 29 per imparare ad usare bene il registratore per memorizzare e caricare i programmi, e anche il 30, se si possiede la stampante, per poterla sfruttare subito. Si ragiona meglio su un listato di programma su carta, che non su un listato solo su video.

Alla fine del libro si hanno le appendici. Si raccomanda di leggere con attenzione l'APPENDICE D, dedicata ai programmi. Questa è stata ampliata rispetto al manuale inglese originale, aggiungendo alcuni programmi e parecchie spiegazioni, che penso saranno gradite ai lettori. I principianti possono imparare molto da un'attenta lettura dei programmi che già funzionano. Le annotazioni possono far

capire cose che altrimenti sfuggirebbero; si può provare a modificare qualcosa (a ragion veduta) e controllare cosa succede. I programmi cercano di mettere in luce le molte possibilità dello ZX Spectrum, pur senza esaurire l'argomento. Penso che saranno particolarmente graditi gli esempi che riguardano i file di dati e i file di byte.

È stata aggiunta l'APPENDICE F per riassumere ad uso dei conoscitori dello ZX 81 le differenze del nuovo calcolatore rispetto al vecchio.

Per finire, qualche cenno sull'implementazione del Basic disponibile sullo Spectrum. Si tratta di una versione molto potente; praticamente, rispetto alla media dei piccoli calcolatori in commercio, manca solo ELSE nella IF e mancano le funzioni di stringa LEFT$, RIGHT$ e MID$, ma esse sono egregiamente sostituite dalla funzione di SLICING che consente di ottenere molto più comodamente le stesse prestazioni e altre in più.

Si possono gestire i file molto bene, naturalmente con programmi di un buon livello. In seguito si potranno gestire anche i microdischi e altri tipi di periferiche. Queste ultime cose non le ho potute ancora materialmente sperimentare, dato che non sono ancora giunte in Italia le apparecchiature, ma non ho alcuna ragione di dubitare della loro validità. Quel molto che è già disponibile può solo far ben sperare per il resto.

La tastiera ha dei veri tasti, all'inizio sembra un pò complessa, dato che ogni tasto può svolgere più funzioni, ma basta poco esercizio per acquistarne padronanza. Si ha la funzione di REPEAT su tutti i tasti.

Il comando di INPUT è molto versatile e consente cose che spesso si desiderebbe poter fare.

Si possono scrivere più istruzioni sulla stessa linea Basic usando il separatore ":". Si dispone dello stesso Editing dello ZX 81, che aiuta a non fare errori di sintassi quando si caricano i programmi. Lo scrolling del video è automatico, ma può essere controllato.

Non posso non spendere qualche parola sulla ricchezza dello Spectrum nella grafica a colori di cui dispone. Ho dedicato alcune serate alla preparazione dei programmi e mi sono trovata molto bene. Oltre ai 10 caratteri grafici disponibili più i 10 in campo inverso, l'utente può facilmente definirne altri 21 e ottenerli premendo un tasto! Sono disponibili funzioni potenti come DRAW e CIRCLE.

Un'altra cosa che vale la pena di menzionare è che risulta molto più tranquilla la gestione del registratore a nastri. Quello che compare sul video durante le operazioni aiuta a capire bene cosa sta succedendo, e il comando VERIFY risulta prezioso per verificare la bontà della registrazione effettuata. È disponibile il sempre desiderato comando MERGE!

Non mi resta altro che augurare a tutti di ricavare molte soddisfazioni dal loro ZX SPECTRUM. Si tratta di un mezzo mediante il quale si può imparare a programmare bene sia in Basic che in linguaggio macchina e che consente di comprendere come funziona un sistema operativo, spianando la strada di accesso all'informatica.

CARATTERISTICHE TECNICHE DELLO ZX SPECTRUM

- ROM 16K usata per Sistema Operativo e Basic.
- RAM 16K (espandibili con +32K) o 48K, di cui: 6912 byte per gestione video (8x8 byte per ognuna delle 768 posizioni carattere + 768 byte per gli attributi); 256 byte per buffer printer; 182 byte per variabili del sistema.
- Video con 24 linee di 32 caratteri, le linee 22 e 23 usate per i comandi e le linee da 0 a 21 a disposizione del programma. Scrolling automatico e controllabile. Risoluzione grafica 256x176 pixel singolarmente indirizzabili.
- 8 colori disponibili per sfondo. bordo e scrittura, controllo lucentezza, intensità e lampeggio.
- BEEP sonoro che può coprire più di 10 ottave.
- Tastiera con 40 tasti di gomma. Repeat automatico su tutti i tasti. Tutte le funzioni sono leggibili sulla tastiera.
- Set caratteri ASCII con minuscole e maiuscole, 20 caratteri grafici programmati e 21 programmabili.
- Collegamento con ZX Printer, 32 caratteri per linea, 9 linee per pollice, 50 caratteri al secondo.
- Annunciata interfaccia RS232.
- Annunciati microdischi da 100K byte con possibilità di collegamenti multipli.
- Collegamento diretto al registratore a nastro; trasferimento a 1500 baud. In 100 secondi trasferisce 16K.
- BASIC esteso con possibilità di gestire file su cassetta e su floppy.

CAPITOLO 1

Il Calcolatore e la sua installazione

La prima parte di questo manuale è stata scritta per due tipi di persone. Soprattutto per chi non sa niente, o quasi niente, sui calcolatori, ma anche per chi ha familiarità con i calcolatori ma vuole sapere con che tipo di calcolatore ha a che fare prima di inserire la spina.

Si consiglia di non leggere la parte relativa al linguaggio BASIC fino a quando non si abbiano assimilati bene i primi capitoli.

Aprendo la scatola dello ZX Spectrum, avete trovato:

1) Il calcolatore. Sul calcolatore ci sono tre prese jack (segnate 9V DC IN, EAR e MIC), una presa per il cavo della televisione e un connettore multipista per collegare altre periferiche. Notate che non ci sono interruttori, per accenderlo basta inserire lo spinotto dell'alimentatore.

2) L'alimentatore che fornisce 9 V in corrente continua non stabilizzata a 1,2 A. Se volete usare un altro alimentatore curate che abbia le stesse caratteristiche.

3) Un cavetto schermato televisivo lungo circa 2 metri per collegare il calcolatore alla televisione.

4) Una coppia di cavetti lunghi circa 75 cm con dei jack da 3,5 mm a ogni capo per collegare il calcolatore al registratore a cassette.

Per poter lavorare con lo ZX Spectrum sarà necessaria una televisione in grado di ricevere la banda UHF, preferibilmente a colori (tipo PAL), dato che lo ZX Spectrum è in grado di fornire immagini a colori. Un televisore in bianco e nero riprodurrà i sei colori dello ZX Spectrum in sei diverse tonalità di grigio.

I componenti del sistema devono essere collegati come segue:

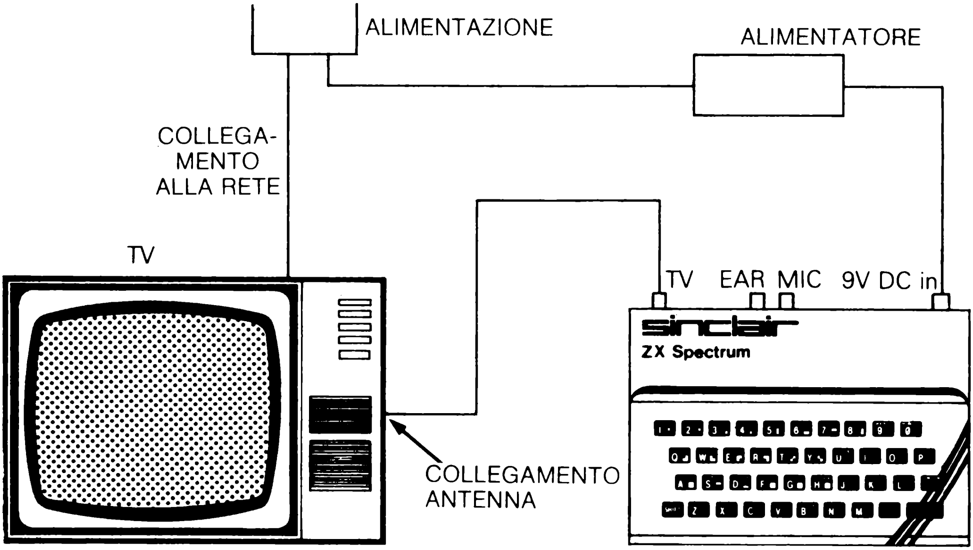


Figura 1

Se la vostra televisione ha due entrate per antenna, collegatevi all'entrata UHF tramite un'adattatore di impedenza.

Terminati i collegamenti accendete l'alimentatore e il televisore. Come prima cosa sintonizzate la televisione sul canale 36 UHF, utilizzato dallo ZX Spectrum. Dovreste vedere la seguente immagine:

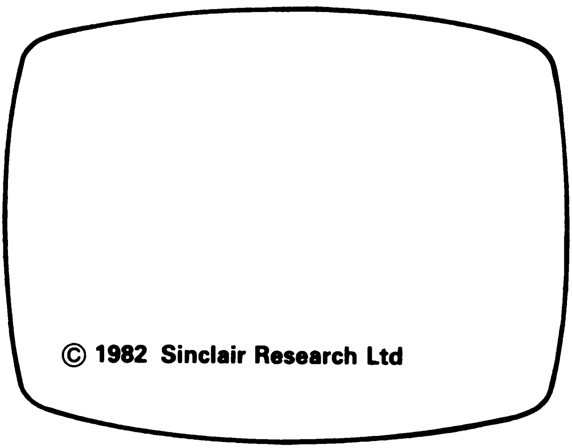


Figura 2

Ricordatevi di azzerare il volume del televisore. Se il vostro televisore ha una sola manopola per tutte le stazioni, per sintonizzarvi dovrete semplicemente girarla fino ad ottenere l'immagine in figura 2; se invece avete un bottone per ogni stazione sceglietene uno non utilizzato e sintonizzatelo.

Quando spegnete lo ZX Spectrum tutte le informazioni introdotte vanno perse. Per memorizzarle in modo permanente potrete usare un registratore a cassette che vi permetterà anche di caricare i programmi che potrete comprare su cassetta. Per collegare il registratore a cassette usate i due cavettini con i jack come spiegato nel capitolo 8.

Appena installato, il calcolatore è pronto per essere usato. In seguito questo libro vi dirà come fare. Se avete già provato a premere qualche tasto, avrete notato che fa cambiare la scritta sul video: ciò è normale, **è impossibile danneggiare il calcolatore in questo modo.** Non abbiate paura di provar qualunque cosa facciate potrete sempre ritornare alle condizioni iniziali, semplicemente sconnettendo per qualche secondo l'alimentazione (spinotto marcato 9 V DC IN). Attenzione però perche così cancellate proprio tutto.

ATTENZIONE Non provate a usare con lo ZX Spectrum l'espansione da 16K RAM per lo ZX 80 e 81 perchè non funziona.

CAPITOLO 2

La Tastiera

La tastiera dello Spectrum è simile a quella di una macchina da scrivere, infatti le lettere e i numeri sono nella stessa posizione, ad eccezione della Q e della Z che sono scambiate di posto e della M che è di fianco alla N invece che alla L, secondo gli standard inglesi. La cosa più importante da osservare è che ogni tasto può avere fino a sei significati diversi a seconda del modo in cui si trova il calcolatore e di quali tasti di controllo (**CAPS SHIFT** e **SYMBOL SHIFT**) sono premuti. Le lettere maiuscole si ottengono premendo insieme al tasto della lettera **CAPS SHIFT**.

Il modo in cui si trova il calcolatore è indicato da una lettera in campo inverso (bianca su fondo nero) che appare sul video e lampeggia. Questo quadratino lampeggiante è il cursore; è importante che impariate a identificarlo in fretta e a distinguerlo da quello che appare sul video perchè tutto quello che scriverete apparirà prima di esso.

Appena acceso il calcolatore sul video appare il messaggio di copyright e, premendo un qualsiasi tasto appare al suo posto la parola scritta sul tasto premuto, sotto la lettera. Queste parole sono le parole chiave del linguaggio BASIC (keyword, in inglese). I tasti producono queste parole chiave quando il calcolatore è nel modo **K**. Notate che non si può scrivere una keyword battendola per intero. Nello ZX Spectrum, diversamente da molti altri calcolatori, esse si introducono con una sola battuta.

Per esempio, se premete la **P** subito dopo l'accensione, sul video apparirà **PRINT**. Dopo **PRINT** scrivete le virgolette ". Osservate che sono in rosso sul tasto **P**. Osservate inoltre che nel cursore appare una **L**, come in figura 3, che significa che il

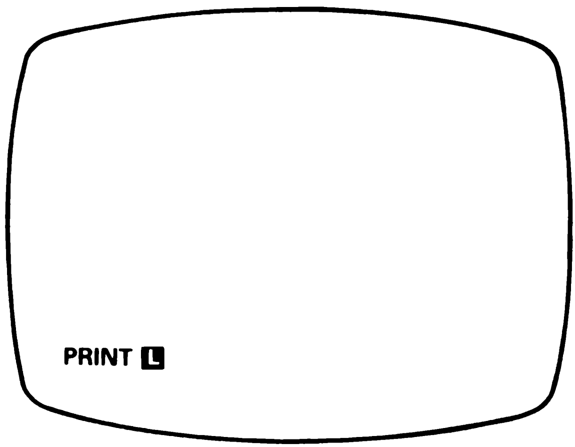


Figura 3 L

calcolatore si aspetta una lettera, o un carattere alfanumerico. Per ottenere le virgolette premete **SYMBOL SHIFT** e, tenendolo premuto, battete **P**.

Osservate che il cursore è ancora nello stato **L**. Scrivete ora la parola "**Ciao**" come su una macchina da scrivere, poi introducete ancora le virgolette. Usate il tasto **CAPS SHIFT** per ottenere la C; con questo tasto, come avrete notato, si ottengono i simboli bianchi sopra i tasti, mentre i simboli rossi si ottengono con **SYMBOL SHIFT**.

Avete appena introdotto una linea di comando, che ordina al calcolatore di stampare **Ciao** sul video. Per rendela esecutiva premete il tasto **ENTER**. Provate, ed in cima al video apparirà la parola

Ciao

e qualche altro carattere (se appare un punto interrogativo lampeggiante, significa che avete commesso un errore: ricominciate da capo). Se è andato tutto bene apparirà un messaggio sulla linea bassa del video che indica cosa è veramente successo all'interno del calcolatore. Sarà di grande importanza quando farete girare dei programmi, per il momento ignoratelo.

Notate anche che la lettera O ed il numero 0 sono visualizzati in modo differente. È importante che vi ricordiate che il numero 0 è sempre barrato. Non usate MAI, come su certe macchine da scrivere, la lettera O come 0 e la lettera L minuscola come 1. Nel manuale non sono stati barrati gli zeri dato che sono chiaramente distinguibili dalle lettere O.

Oltre al modo **K** ed al modo L che avete appena incontrato, il calcolatore si può trovare in altri modi. Se volete introdurre un testo tutto maiuscolo senza tenere sempre premuto **CAPS SHIFT**, potete passare al modo C, usando la funzione **CAPS LOCK**: poichè **CAPS LOCK** si trova scritta in bianco sopra il **2**, per attivarla premete **2** contemporaneamente a **CAPS SHIFT**: una **C** lampeggiante apparirà nel cursore. Per ritornare al modo **L**, premete semplicemente **CAPS LOCK** una seconda volta. Osservate che se passate al modo **C** quando non siete nel modo **L** sembra che non succeda niente. Solo quando il calcolatore vi richiederà delle lettere, vedrete apparire **C** al posto di **L**.

Sulla tastiera dello ZX Spectrum ci sono anche otto caratteri grafici situati sui tasti da **1** a **8**. Possono essere usati insieme a lettere e numeri, ma per introdurli è necessario che il calcolatore sia nel modo grafico. Notate che **GRAPHICS** è scritto sopra il tasto **9** attivatelo allo stesso modo di **CAPS LOCK**: nel cursore apparirà una **G**. Per ritornare al modo L premete il tasto **9** insieme a **CAPS SHIFT**.

Ultimo, ma non meno importante, il modo **E**, (modo ESTESO) permette di usare le funzioni scientifiche e di programmazione più avanzate. Si attiva e disattiva premendo contemporaneamente **CAPS SHIFT** e **SYMBOL SHIFT**.

È importante che sappiate come correggere gli inevitabili errori di battuta senza dover ricominciare tutto da capo. Il **DELETE** (sopra il tasto **0**) permette di cancellare l'ultimo carattere prima del cursore. Ritorniamo all'esempio della **PRINT** supponiamo invece di:

PRINT "Ciao"

di avere scritto per errore

PRINT pCiao"

cioè di avere dimenticato di premere **SYMBOL SHIFT** per ottenere le virgolette. Il calcolatore non accetta i comandi errati e li segnala con un punto di domanda lampeggiante alla fine della riga. Provate.

Per correggere l'errore, dovrete posizionare il cursore dopo il carattere errato, in questo caso con la freccia verso sinistra (sopra il **5**). Osservate che se tenete premuto il tasto per più di tre secondi, comincerà a ripetersi automaticamente con un cicalino. Quando siete subito dopo la **P**, cancellatela con il **DELETE (CAPS SHIFT e 0)** e poi introducete le virgolette correttamente. Provate anche la freccia verso destra, tanto per abituarvi. Premete **ENTER** e, se non avete interrotto il lavoro, vedrete apparire un altro **Ciao** sotto il primo **Ciao**.

Ogni volta che commettete un errore di battitura, correggetelo come spiegato. Ricordatevi però che non si può scrivere direttamente sopra i caratteri errati, bisogna cancellarli e rimpiazzarli.

Troverete una descrizione completa della tastiera nel primo capitolo sul BASIC.

CAPITOLO 3

Numeri, lettere e il computer come calcolatrice

Avete già visto come con la **PRINT** si possa ottenere la stampa di lettere e simboli alfanumerici, e come si rendono i comandi esecutivi con **ENTER**. D'ora in poi si considererà sottointesa l'operazione di **ENTER**.

Nel capitolo precedente avete usato la **PRINT** per scrivere dei messaggi tra virgolette; ora si esaminerà l'uso della stessa istruzione con i numeri.

Se introducete

PRINT 2

apparirà il numero **2** sullo schermo.

Si può anche stampare insieme lettere e numeri; ad esempio:

PRINT 2,"ABC"

Notate che c'è una virgola tra **2** e **"ABC"**.

Provate ora

PRINT 2; "ABC"

e poi

PRINT 2"ABC"

Nel primo caso la virgola ha prodotto una separazione di 16 colonne, nel secondo caso il punto e virgola non ha prodotto alcuna spaziatura, nel terzo si è avuto errore.

La **PRINT** può essere anche usata direttamente con le funzioni matematiche della tastiera, permettendo di usare lo ZX Spectrum come una calcolatrice elettronica.

per esempio:

PRINT 2+2

stampa il risultato dell'operazione,

diversamente da

PRINT "2+2"

È possibile combinare più utilmente queste due istruzioni:

PRINT "2+2="; 2+2

Provate anche:

PRINT 3–2
PRINT 4/5
PRINT 12*2

Il simbolo / indica la divisione ed il simbolo * indica la moltiplicazione. Notate che si usa l'asterisco per evitare confusioni con la **x**.

Provate a fare calcoli anche con numeri decimali e negativi.

Se fate abbastanza prove da riempire le 22 righe della parte superiore del video, noterete che la prima riga sparisce e tutte le altre si spostano di una posizione per far posto ad una nuova riga. Questo meccanismo si chiama *scrolling.*

Lo ZX Spectrum esegue le operazioni aritmetiche nel normale ordine algebrico. Prima gli elevamenti a potenza e le radici, poi le moltiplicazioni e le divisioni e per ultime le addizioni e le sottrazioni. Per questa ragione si dice che la moltiplicazione ha priorità superiore alla sottrazione e uguale alla divisione. Le operazioni con la stessa priorità vengono eseguite in ordine da sinistra a destra. Per esempio:

PRINT 2+3*5

dà per risultato 17, perchè 2 viene sommato al risultato di 3 per 5. Analogamente:

PRINT 20-2*9+4/2*3

esegue le operazioni nel seguente ordine:prima 2*9, poi 4/2 e moltiplica il risultato per 3, ottenendo così 20–18+6, quindi esegue somme e sottrazioni da sinistra verso destra, dando come risultato 8, come mostrato in tabella.

i	20–2*9+4/2*3	Prima vengono eseguite le moltiplicazioni e le divisioni in ordine da sinistra verso destra
ii	20–18+4/2*3	
iii	20–18+2*3	
iv	20–18+6	e poi le addizioni e le sottrazioni
v	2+6	
vi	8	

Tenete ben presente la priorità delle operazioni. Il calcolatore ne tiene conto attribuendo ad ogni operazione un diverso valore di priorità da 1 a 16, eseguendo

prima quello di priorità più elevata: * e / hanno priorità 8, + e − hanno priorità 6.
L'ordine di calcolo è rigoroso, ma potete usare le parentesi per eseguire le operazioni in un altro ordine: qualunque cosa racchiusa tra parentesi viene calcolata prima, secondo le solite regole, e poi considerata come un singolo numero. Quindi

PRINT 3*2+2

dà per risultato 6+2=8; ma

PRINT 3*(2+2)

dà per risultato 3*4=12.

Quando il calcolatore richiede un numero, può essere utile invece utilizzare un'espressione di questo tipo, che gli permette di ricavare da solo il valore di cui ha bisogno.
Si possono usare anche numeri decimali. Fate bene attenzione che, differentemente dalle nostre abitudini, il calcolatore richiede il punto **.** (invece della virgola) per separare la parte intera dai decimali. Come la maggior parte delle calcolatrici tascabili, lo ZX Spectrum lavora anche con la notazione esponenziale, in cui il numero è seguito da un esponente formato dalla lettera **e**, seguita da un numero che può essere anche negativo. L'esponente sposta il punto decimale (che siamo abituati a chiamare virgola) di tante posizioni verso destra, ovvero moltiplica per 10, quant'è il suo valore; verso sinistra se è negativo. Per esempio:

2.34e0=2.34
2.34e3=2340
2.34e−2=0.0234 e così via.

verificate questi esempi sul calcolatore. Questo è uno dei pochi casi in cui non potete sostituire un numero con un'espressione. Per esempio non potete scrivere

(1.34+1)e(6/2).

Esistono anche espressioni non numeriche, chiamate stringhe, formate da lettere ed altri caratteri. Le avete già incontrate quando avete provato a stampare una parola con le virgolette. In effetti assomigliano moltissimo alla forma più semplice delle espressioni numeriche formate da un solo numero. Il + è l'unico operatore che può essere usato anche per le stringhe: semplicemente le unisce una di seguito all'altra. Provate

PRINT "calcola"+"tore".

Potete unire quante stringhe volete con una sola espressione, usando anche le parentesi, che non hanno però alcun effetto pratico.

CAPITOLO

4

Alcuni comandi elementari

La memoria del calcolatore può essere usata per memorizzare qualunque tipo di dati: con la **PRINT** avete visto stampare lettere e numeri. Se volete dire al calcolatore di memorizzare un numero o una stringa alfanumerica dovete prima riservargli un posto nella memoria. Molte calcolatrici tascabili hanno un tasto per memorizzare un numero; il calcolatore è molto più capace: ha moltissime posizioni di memoria predisposte a ricevere numeri.

Ad esempio, supponete di voler ricordare la vostra età. L'istruzione **LET** (tasto L) servirà allo scopo.

LET anni=34

In questo modo il numero 34 viene messo in un'area di memoria che ha per nome "anni" per farlo stampare basterà scrivere:

PRINT anni

È facilissimo modificare il contenuto di "anni". Scrivete:

LET anni=56

PRINT anni

Leggerete 56 sullo schermo. *anni* è un esempio di *variabile*, chiamata così dato che il suo valore può variare. Scrivete:

PRINT "La vostra età è ";anni

Una variabile può anche contenere una stringa alfanumerica: in questo caso il suo nome deve essere seguito dal dollaro $. Ci sono quindi due tipi di variabili: le variabili numeriche e le variabili alfanumeriche. Ora sapete che **a$** è una variabile alfanumerica. Potete assegnarle il valore **"La vostra età è"** scrivendo:

LET a$="La vostra età è"

e stamparla con

PRINT a$

provate. Vi riapparirà la scritta.

È importante ricordare che i nomi delle variabili alfanumeriche possono essere formati da una sola lettera oltre al dollaro. Se non avete ancora spento il calcolatore provate:

PRINT a$; anni

ma cercate di prevedere cosa succede.

Per assegnare dei valori alle variabili potete usare **INPUT** oltre a **LET**. Con essa i valori delle variabili vi saranno richiesti durante l'esecuzione del programma. Scrivete:

INPUT anni

Premete **ENTER**: sullo schermo apparirà il cursore con la **L** : il calcolatore richiede un'informazione. Scrivete la vostra età e premete **ENTER**. Anche se non avete notato alcun cambiamento, l'assegnazione è stata eseguita. Per verificare scrivete:

PRINT anni

Unite queste istruzioni:

LET b$="Quanti anni avete?"
LET a$="La vostra età è "
INPUT(b$);anni:PRINT a$; anni

Notate che l'ultima riga è formata da due istruzioni, separate dai :.

INPUT(b$);anni

è in questo caso equivalente a:

INPUT "Quanti anni avete? ";anni

CAPITOLO 5

Programmazione elementare

Finora avete dato al calcolatore solo comandi in modo immediato, ma vi sarete accorti che, anche combinandoli insieme, non si possono fare grandi cose.

Ben altra è la potenza degli stessi comandi in un programma! Un programma è una sequenza ordinata di istruzioni elementari, che, scritta una volta per tutte, mette in grado il calcolatore di risolvere un certo problema, di svolgere una funzione, di fare insomma un'operazione predefinita.

I progammi possono essere scritti in differenti linguaggi. Il linguaggio è il mezzo con cui il programmatore può comunicare alla macchina quello che deve fare. Sfortunatamente gli esseri umani ed i calcolatori non parlano la stessa lingua, è stato quindi necessario ideare diversi metodi per comunicare. Alcuni sono molto naturali per il computer, ma difficili per gli uomini e viceversa. Così i linguaggi più "umani", detti ad alto livello, devono essere interpretati o tradotti.

Il vostro ZX Spectrum utilizza uno di tali linguaggi, il BASIC.

BASIC significa in italiano "codice simbolico multi-uso di istruzioni per principianti", dall'inglese Beginners All-purpose Symbolic Instruction Code. È stato realizzato all'università di Dartmouth, New Hampshire, USA, nel 1964. È di gran lunga il linguaggio più comune sui personal computers, e appunto per questo ne sono state scritte moltissime versioni, che pur essendo uguali nella sostanza, differiscono in qualche dettaglio. Ecco perchè questo manuale è stato scritto appositamente per lo ZX Spectrum. Tuttavia, questo BASIC, è molto vicino ad un teorico modello universale, per cui non avrete difficoltà ad adattare programmi BASIC di altre macchine per lo ZX Spectrum. Ad esempio, il BASIC dello ZX Spectrum, non permette, a differenza della maggior parte degli altri sistemi, di omettere l'istruzione **LET** davanti ad una assegnazione di valore ad una variabile.

C'è un limite al numero di istruzioni che possono essere contenute nella memoria di un calcolatore e lo ZX Spectrum vi avverte quando viene raggiunto questo limite emettendo un suono.

Scrivendo dei programmi in BASIC, è necessario comunicare al calcolatore anche l'ordine in cui le istruzioni devono essere eseguite. Per questo è obbligatorio cominciare ogni linea di un programma con un numero, chiamato numero di linea.

È l'ordine dei numeri e non l'ordine di inserimento a determinare la sequenza di esecuzione delle istruzioni. Come numeri di linea si possono usare solo numeri interi positivi; così per rendere possibili eventuali inserimenti di linee, è preferibile numerare le istruzioni di 10 in 10. Partendo da 10: 10, 20, 30, ecc.

Considerate la serie di comandi alla fine del capitolo precedente. Se volete ripeterla dovete riscrivere i comandi per intero ogni volta. Se fate un programma, lo potrete usare quante volte vorrete.

Scrivete ora, usando **ENTER** alla fine di ogni linea.

```
10 LET b$="Quanti anni avete?"
20 LET a$="La vostra età è "
30 INPUT (b$);anni
40 PRINT a$;anni
```

Notate che non è necessario battere gli spazi, salvo che fra le virgolette.

Il calcolatore adesso ha memorizzato il programma: non accadrà nient'altro fin quando non gli direte di eseguirlo. La parola chiave per eseguire un programma è la **RUN** (tasto **R**). Provate.



Figura 4

Forse avete notato una freccia verso destra dopo ogni linea introdotta. Indica appunto che la linea è stata introdotta. Se volete rivedere la lista del programma, premete ancora **ENTER** oppure **LIST**. Con **RUN** potete eseguire il programma quante volte volete. Quando non avete più bisogno di questo programma, lo potete cancellare con il comando **NEW** che prepara la memoria pulita per un altro programma. Ogni comando deve essere seguito da **ENTER**.

Battete **NEW** e poi **LIST** e guardate cosa succede.

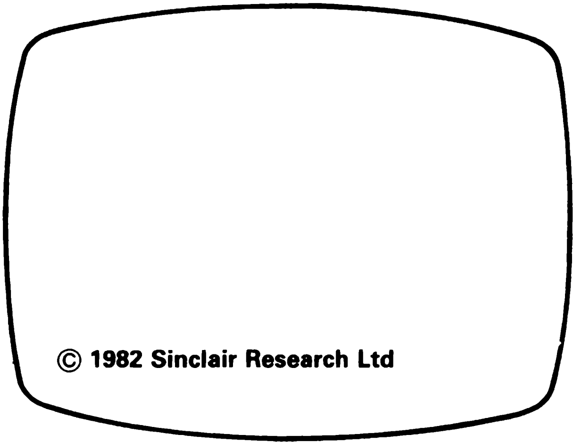


Figura 5

Per ricapitolare: quando introducete un comando preceduto da un numero, significa che non è un comando semplice, ma una linea di programma, che non viene eseguita ma memorizzata per essere usata più tardi, quando l'operatore lo richiederà col comando **RUN**.

Preparatevi per un altro programma un pò più matematico, che stamperà il quadrato di tutti i numeri da 1 a 10 (il quadrato di un numero è il numero moltiplicato per sè stesso).

Generando i numeri da 1 a 10, incontrerete un altro concetto ricorrente nella programmazione in BASIC: un metodo con cui il calcolatore può contare. Prima avete visto che un numero può essere conservato nella memoria del calcolatore associandolo ad un nome, o più tecnicamente, assegnando un valore ad una variabile. Ora vedrete come con il comando **FOR...TO...STEP** si può assegnare alla variabile **x** il valore 1 ed incrementarlo di 1 in 1 fino a 10.

Pulite la memoria (**NEW**) ed introducete il seguente programma:

10 FOR x=1 TO 10 STEP 1

Se si conta a passi di 1, lo **STEP** può essere omesso. La prossima linea deve dire al calcolatore cosa fare con la **x** per ogni valore che assume.

20 PRINT x,x*x

Infine è necessaria una linea per dire al calcolatore di continuare col successivo valore di **x**.

30 NEXT x

Con quest'ultima istruzione il calcolatore ritorna alla linea 10 e ripete la sequenza. Quando **x** supera 10 il calcolatore prosegue, per esempio, con un'eventuale linea 40.

Il programma deve apparire sullo schermo come segue:

```
10 FOR x = 1 TO 10 STEP 1
20 PRINT x,x*x
30 NEXT x
```

Per completezza è necessaria un'altra linea per dire al calcolatore che il programma è finito quando **x**=10. Terminate il programma con:

```
40 STOP
```

Fate girare il programma e vedrete apparire due colonne, la prima con i valori di **x**, la seconda con i valori del quadrato di **x**. È possibile mettere un nome ad ogni colonna aggiungendo un'altra linea. Provate:

```
5 PRINT "x","x*x"
```

Notate che nonostante questa linea sia stata introdotta dopo tutte le altre, il calcolatore la mette automaticamente al primo posto, basandosi sul numero di linea.

Provate ora a scrivere altri semplici programmi con altre funzioni matematiche. Se avete qualche dubbio consultate le opportune pagine del BASIC.

CAPITOLO 6

Uso del registratore a cassette

È piuttosto noioso dover introdurre un programma nel calcolatore ogni volta che lo si vuole usare. Lo ZX Spectrum offre la possibilità di registrare i programmi su nastro magnetico con un qualunque registratore a cassette. Se avete un programma in memoria provate a registrarlo con la procedura descritta più avanti.

Appena sarete in grado di registrare programmi su cassetta, portete ricaricarli quante volte vorrete.

La maggior parte dei registratori a cassette è adatta; infatti i più economici registratori portatili monofonici sono, per il calcolatore, allo stesso livello dei più costosi; sarà molto utile un contagiri.

Il registratore deve avere una presa di ingresso per il microfono, e una presa d'uscita per le cuffie (se quest'ultima manca cercate la presa per l'altoparlante esterno). Le prese dovranno essere del tipo jack da 3,5 mm, come i cavetti in dotazione. Non usate altre uscite perchè potrebbero dare segnali inadatti.

È consigliabile usare cassette a basso rumore. Collegate il registratore al calcolatore mediante i cavetti in dotazione: uno di questi servirà a collegare la presa per il microfono del registratore alla presa marcata MIC sul retro della macchina, l'altro servirà a collegare la presa per la cuffia del registratore alla presa marcata EAR. Anche se doveste collegare i cavi in modo errato non rischiate di danneggiare lo ZX Spectrum.

Quando usate la **SAVE** per registrare un programma su nastro assicuratevi che il cavetto che collega la presa EAR al registratore sia scollegato almeno da una parte. Altrimenti non otterrete altro che una nota continua ed inutile sul nastro. Questo perchè quando il registratore registra amplifica il segnale in entrata sulla presa MIC e lo restituisce sull'uscita EAR. Se il segnale in uscita torna al calcolatore si avrà una oscillazione continua che rovinerà il segnale che volevate registrare.

Prima di memorizzare un programma su nastro, dovrete dargli un nome. Ad esempio potete chiamare il programma dei quadrati appunto "Quadrati" specificando il nome dopo **SAVE** così:

SAVE "Quadrati"

Il nome non può avere più di dieci caratteri, che possono essere solo lettere e numeri.

Il calcolatore risponderà al messaggio precedente con la scritta **Start tape then press any key**, che significa: avviate il nastro e poi premete un qualunque tasto. Per capire cosa succede premete un tasto senza avviare il registratore. Nei bordi dello schermo osserverete durante i primi 5 secondi delle strisce blu pallido e rosse che si muovono lentamente verso l'alto, poi delle righe blu e gialle seguite da un momento di normalità. Ancora per un paio di secondi le strisce blu pallido e rosse, infine ancora le righe blu e gialle. Tutto ciò rivela che l'informazione è memorizzata in due blocchi, separati da uno spazio muto, l'inizio di ognuno dei quali è rappresentato

dalle strisce blu pallido e rosse, e il cui contenuto corrisponde alle righe blu e gialle. Il primo è un blocco preliminare, contenente il nome e qualche altra informazione sul programma; il secondo contiene il programma e le sue variabili. La sezione bianca rappresenta la demarcazione tra i due blocchi.

Ora procedete pure alla registrazione su cassetta seguendo le istruzioni:

1. Posizionate il nastro in una parte libera, oppure in una zona che desiderate riincidere.

2. Scrivete

SAVE "Quadrati" (ed **ENTER**)

3. Avviate il registratore col tasto di registrazione premuto (REC).

4. Premete un carattere qualunque dello ZX Spectrum.

5. Osservate lo schermo: quando apparirà il messaggio **0 OK** il calcolatore avrà terminato, fermate il registratore.

Se volete assicurarvi che l'operazione è stata compiuta, potete controllare il segnale inciso sul nastro per mezzo dell'istruzione **VERIFY**, così:

1. Regolate il volume del registratore circa a metà e ricollegate il cavo "EAR".

2. Riavvolgete la cassetta sino al punto in cui avevate iniziato la registrazione.

3. Scrivete:

VERIFY "Quadrati"

(La **VERIFY** è **R** shiftata nel modo esteso).

4. Avviate il nastro col tasto d'ascolto (PLAY).

Sul bordo dello schermo si alterneranno i colori rosso e blu pallido, fino al momento in cui la testina del registratore raggiungerà l'inizio della registrazione; successivamente comparirà la stessa sequenza di colori osservata durante la registrazione. In corrispondenza alla fase in cui i bordi sono normali apparirà la scritta **Program Quadrati**, dato che durante una ricerca il calcolatore riporta sullo schermo tutto quello che trova sul nastro. Al termine delle sequenze dei colori, se il programma è memorizzato correttamente, apparirà il messaggio **0 OK**: ora siete completamente sicuri che potrete sempre ricaricare il vostro programma. Se avete notato qualche errore, leggete attentamente i paragrafi che seguono.

Assicuratevi che il vostro programma sia registrato

Se non avete visto comparire il nome del programma, significa che quest'ultimo non è stato registrato o richiamato correttamente. Per capire in quale delle due fasi si è avuto l'errore, dovete riavvolgere il nastro fino all'inizio della registrazione ed ascoltare il contenuto del nastro direttamente dall'altoparlante del registratore (dovrete probabilmente scollegare il cavetto dalla presa per le cuffie del registratore). In corrispondenza dell'apparizione sullo schermo dei colori rosso e blu chiaro (pallido) sentirete una nota continua e pulita; con i colori blu e giallo, sentirete invece un suono più sgradevole, simile ai "bip" del codice morse, o a una radio disturbata da un uragano; i due suoni sono abbastanza forti da coprire una conversazione se il volume del registratore è al massimo.

Se non li avete uditi, probabilmente il programma non è stato registrato. Controllate in questo caso se i collegamenti sono giusti: verificate che le prese "MIC" siano collegate, e che non lo siano le "EAR". Capita che in alcuni registratori gli spinotti jack non facciano contatto se sono infilati fino in fondo nella presa; provate quindi a estrarli di circa due o tre millimetri, oppure fino a quando non sentirete che lo spinotto sia sistemato più comodamente. Inoltre controllate se non avete tentato di registrare sulla parte iniziale di nastro della cassetta, che è plastica inservibile: dopo tutte queste verifiche, riprovate a registrare il programma.

Se invece avete udito i suoni, allora l'errore è da ricercare nella fase di richiamo.

Controllate di nuovo i cavetti ed il volume; se quest'ultimo è troppo basso, il calcolatore potrebbe non aver ricevuto correttamente il segnale, e quindi causato sequenze errate sullo schermo; se invece è troppo alto, il calcolatore potrebbe aver ricevuto un segnale distorto, avvertibile anche dall'altoparlante. Comunque fra i due estremi vi è un largo campo di volumi accettabili, quindi provate a regolare il volume.

È anche possibile che il calcolatore trovi il programma e ne scriva il nome, ma non riesca a caricarlo. La ragione può essere un errore di battitura nel nome: il calcolatore ignorerà il programma registrato, trovandolo con un nome differente da quello cercato e continuerà a mostrare i colori rosso e blu pallido.

Un'altra ragione può essere un vero e proprio errore di registrazione: ma in questo caso il calcolatore scriverà il messaggio **R Tape loading error** sullo schermo, che indica un errore durante la procedura di **VERIFY**; provate a registrarlo di nuovo.

Inoltre il volume può non essere ancora regolato alla perfezione, ma ciò non dovrebbe causare problemi, dato che il calcolatore riesce a leggere il primo blocco.

Ora supponete di aver registrato e verificato il programma: potete caricarlo con la stessa procedura, ma scrivendo

LOAD "Quadrati"

invece di

VERIFY "Quadrati"

LOAD è sul carattere **J** della tastiera. Dal momento in cui **LOAD** viene eseguita correttamente, siete sicuri di non avere più problemi di caricamento.

La **LOAD** cancella il vecchio programma (e le sue variabili) nel calcolatore, prima di caricarne uno nuovo.

Terminata la fase di caricamento di un programma, l'istruzione **RUN** farà svolgere, o più tecnicamente "girare" il programma.

Sono disponibili cassette contenenti programmi preregistrati, ma potrete usare solo quelle preparate apposta per lo ZX Spectrum, altri tipi di calcolatori hanno diverse tecniche di registrazione non compatibili fra loro.

Se il vostro nastro contiene più di un programma per lato, essi devono avere nomi diversi per poterli identificare. Potete scegliere quale programma caricare con la **LOAD**: per esempio, se volete richiamare il programma chiamato "elicottero", dovrete scrivere:

LOAD "elicottero"

(**LOAD** significa *caricare* il primo programma che le testine del registratore trovano: ciò può essere molto utile se non ricordate il nome).

CAPITOLO 7

I colori

Lo ZX Spectrum è in grado di scrivere e disegnare a colori. Lo schermo è diviso in due parti: la parte esterna, che d'ora in poi sarà chiamata **BORDER** (= cornice) e la parte interna che sarà chiamata **PAPER** (= carta). È possibile cambiare tutte le volte che si vuole il colore di queste due zone, sia direttamente dalla tastiera che da programma.

Lo ZX Spectrum usa otto colori che sono numerati da 0 a 7. Su un televisore in bianco e nero i numeri più bassi corrispondono ai grigi più scuri, ed i più alti ai grigi più chiari. Ecco la lista dei colori coi rispettivi codici:

0 nero
1 blu
2 rosso
3 porpora o magenta
4 verde
5 blu pallido o ciano
6 giallo
7 bianco

All'accensione il sistema lavora in bianco e nero quindi con valori di **BORDER** e **PAPER** uguali a 7. Il colore dei caratteri che vengono scritti sullo schermo è definito dal comando **INK** (normalmente 0, il nero). Provate a cambiare i valori iniziali. Scrivete:

BORDER 2

Come premete **ENTER** il bordo diventa rosso. Notate che il bordo comprende l'area in basso nello schermo dove vengono scritti i comandi e le istruzioni. Provate altri valori di **BORDER**.

Usando due colori diversi per **PAPER** e **BORDER** si può vedere come la parte bassa dello schermo invada lentamente l'alta quando si scrivono più di due linee senza **ENTER**.

Provate analogamente a cambiare il colore della zona centrale scrivendo

PAPER 5

Il comando **PAPER** è un comando disponibile nel modo esteso, già menzionato; per ottenerlo è necessario premere contemporaneamente **CAPS SHIFT** e **SYMBOL SHIFT** e quindi la **C** maiuscola (con lo **SHIFT**). Per far cambiare il colore dello schermo a ciano dovete premere **ENTER** due volte. Il primo **ENTER** prende nota del comando **PAPER** e lo cancella, il secondo, causando la ristampa del listato del programma, usa il nuovo colore per la carta modificando il colore del video. Se non avete notato nessun cambiamento sul vostro televisore a colori provate a regolare il controllo dei colori e magari anche la sintonia.

Il comando **INK** è simile al comando **PAPER** solo che controlla il colore dei caratteri sulla sezione **PAPER** dello schermo. Ovviamente se i colori di **INK** e **PAPER** sono uguali non si vede nulla!

I comandi **BORDER**, **PAPER** e **INK** possono essere usati anche nei programmi come in questo esempio che mostra tutti i colori disponibili:

```
10 FOR x=0 TO 7
20 BORDER x
30 PAPER 7–x: CLS
40 PAUSE
50 NEXT x
```

Questo programma mostra gli otto colori facendo risaltare bene il bordo (ricordatevi il **RUN** per farlo girare). Il **CLS** dopo **PAPER** obbliga il calcolatore a pulire il video con il nuovo colore di **PAPER**. Il **PAUSE** ferma il programma per un secondo lasciando il tempo per vedere cosa succede (provate a far girare il programma senza il **PAUSE**). Per vedere come funziona l'**INK** introducete il seguente programma dopo un **NEW**.

```
10 BORDER 7
20 PAPER 1
30 INK 4
40 PRINT "Caratteri verdi su fondo blu"
```

Ci sono altri comandi relativi ai colori dello ZX Spectrum che sono spiegati in dettaglio nei capitoli sul BASIC.

CAPITOLO 8

I suoni

Lo ZX Spectrum può produrre suoni di frequenza e durata a piacere. Per ordinare al calcolatore l'emissione di un suono si usa il comando **BEEP**. **BEEP** è ottenibile col modo esteso con il tasto **Z**.

La frequenza normale di **BEEP** è il DO centrale, ma si può ottenere qualunque nota indicandone la distanza in semitoni o in parti di essi sopra o sotto questo. Se scrivete

BEEP 2,0

il calcolatore emette un DO centrale della durata di due secondi.

Ricordate che il primo numero indica la lunghezza della nota in secondi, il secondo la distanza della nota in semitoni dal DO centrale; dovrà quindi essere 0 per il DO centrale, 1 per il DO#, 2 per il RE, 12 per il DO dell'ottava superiore dato che dodici semitoni formano un'ottava, −1 per il SI sotto, −12 per il DO dell'ottava inferiore e così via.

Provate:

BEEP 1,4: BEEP 1,2: BEEP 2,0

Dovreste sentire la prima battuta di "Three Blind Mice". Usando diversi **BEEP** con i due punti non dovrebbe essere difficile, con un pò di pratica, produrre una melodia. Divertitevi a provare.

Notate che i due punti servono in generale per mettere più istruzioni sulla stessa linea.

Per fare un esempio più complicato potete fare un unico comando usando sia **BEEP** che **BORDER**.

BORDER 1: BEEP 1,14: BORDER 3: BEEP 1,16: BORDER 4: BEEP 1,12: BORDER 6: BEEP 1,0: BORDER 5: BEEP 4,7: BORDER 1

Notate che al calcolatore non importa se una linea eccede la larghezza massima del video: la prosegue sulla riga successiva.

Un breve programma per suonare una serie di note potrebbe essere il seguente:

```
10 FOR x=0 TO 24
20 BEEP 2,x
30 NEXT x
```

Con questo comando si possono fare molte più cose: consultate il relativo capitolo sul BASIC.

CAPITOLO 9

Sotto il coperchio

La fotografia mostra l'interno dello ZX Spectrum.

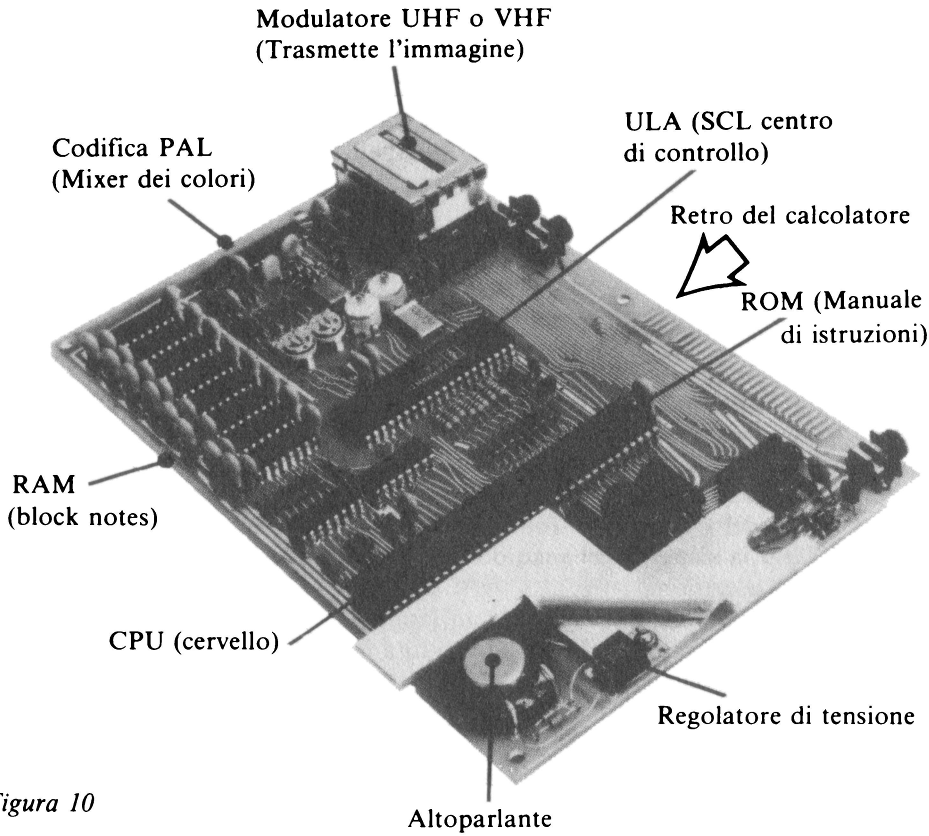


Figura 10

Come potete vedere ogni cosa è indicata con una sigla di tre lettere. I pezzi di plastica nera rettangolari con molte gambettine di metallo sono i circuiti integrati, che in pratica svolgono tutto il lavoro. Dentro ognuno di essi c'è un quadratino di silicio di circa sei millimetri di lato, collegato con dei fili alle gambette di metallo. Su ognuno di questi "chip" di silicio ci sono migliaia di transistors, che formano i circuiti elettronici che *sono* il calcolatore.

Il cervello che controlla le operazioni è il microprocessore, spesso chiamato CPU (Central Processor Unit o unità centrale di processo). Questo in particolare è lo Z80A, che è una versione più veloce del popolare Z80.

Il microprocessore controlla il calcolatore, fa i conti, controlla quali tasti sono stati premuti, decide cosa fare di conseguenza, decide insomma quello che deve fare il calcolatore. Ma per quanto sia intelligente, non può lavorare da solo: per esempio non sa niente sul BASIC o sull'aritmetica decimale, e deve prendere istruzioni da un altro chip, la ROM (Read Only Memory o memoria a sola lettura). La ROM contiene una lunga lista di istruzioni che formano un programma che dice al calcolatore come comportarsi in tutte le possibili situazioni. Questo programma non è scritto in BASIC, ma in quello che è chiamato linguaggio macchina dello Z80, e non è nient'altro che una lunga lista formata da 16384 (16*1024) numeri. Ecco perchè il BASIC dello ZX Spectrum è a volte chiamato BASIC da 16K (1K = 1024).

Per quanto chip del genere siano usati in tutti i calcolatori, questa particolare sequenza di istruzioni è stata fatta appositamente per lo ZX Spectrum, e non andrà bene su nessun altro.

Gli otto chip allineati sono la memoria RAM (Random Access Memory o memoria ad accesso casuale) che lavora in stretta collaborazione con altri due chip. Nella RAM il microprocessore memorizza tutte le informazioni che vuole avere a disposizione, il programma BASIC, le variabili, il quadro televisivo ed altre informazioni che dipendono dallo stato del calcolatore.

L'altro grande chip è l'SCL (Sinclair Computer Logic). Esso è veramente il "centralino" del sistema; controlla che tutti gli ordini del processore vengano eseguiti, legge la memoria del quadro video, manda gli opportuni segnali all'interfaccia per la televisione.

Il codificatore PAL è formato da un gruppo di componenti che convertono l'uscita logica del chip SCL in segnali adatti a una televisione PAL.

Il regolatore di voltaggio stabilizza l'uscita instabile dell'alimentatore a 5 Volts assolutamente costanti.

Con questo è finita la parte introduttiva: se vi sentite pronti leggete la parte sul BASIC.

CAPITOLO 10

Introduzione ai capitoli successivi

Che abbiate letto o meno i capitoli introduttivi, dovreste sapere che i comandi sono eseguiti immediatamente e che le linee di un programma iniziano con un numero di linea e sono memorizzate per essere eseguite in un secondo momento; inoltre dovreste conoscere i comandi **PRINT, LET** ed **INPUT** (che possono essere usati su tutte le macchine che lavorano in BASIC), e **BORDER, PAPER** e **BEEP** (che sono usati sullo ZX Spectrum).

I capitoli seguenti approfondiscono alcuni concetti già esposti; spiegano esattamente cosa dovete e non dovete fare, contengono alcuni esercizi da non tralasciare, dato che sviluppano i concetti spiegati nel testo. Fate quelli che vi interessano maggiormente, o che vi sembrano riguardare qualcosa che non avete ben capito.

Qualunque esercizio svolgiate, continuate ad usare il calcolatore; non chiedetevi: "che cosa farà la macchina se gli ordino questo e quest'altro?". La risposta è semplice: scrivete e vedrete. Ogni volta che il manuale vi consiglia di scrivere qualcosa, vi dovreste chiedere se esiste qualche altro comando alternativo, ed analizzare attentamente le vostre deduzioni. Più programmi sviluppate, meglio capirete la macchina.

Al termine di questo manuale di programmazione vi sono alcune appendici, che includono parti sull'organizzazione della memoria, sul modo in cui il calcolatore lavora con i numeri, ed una serie di programmi che sfruttano le capacità dello ZX Spectrum.

La tastiera

L'insieme dei caratteri dello ZX Spectrum comprende i *simboli semplici* (lettere, numeri, ecc.) ed i *simboli composti* (istruzioni, messaggi dalla tastiera, ecc.) che vengono introdotti con una sola battuta e non scritti per esteso. Per ottenere tutte queste funzioni e comandi, alcuni tasti presentano fino a sei significati diversi, selezionabili premendo il tasto in questione contemporaneamente a **CAPS SHIFT** o a **SYMBOL SHIFT** e cambiando il *modo* in cui si trova la macchina.

Il modo è indicato da una lettera lampeggiante all'interno del *cursore*, che indica il punto dello schermo ove comparirà il primo carattere battuto.

Il modo K (per keyword, in italiano parola chiave) sostituisce automaticamente il modo L quando il calcolatore attende o un comando o una linea di programma (invece che dei dati in **INPUT**). Questo accade o all'inizio della riga, o subito dopo un **THEN** o dopo : (salvo che nelle stringhe). In questo modo se non usate nessuno **SHIFT**, il primo tasto che premerete potrà essere interpretato o come la keyword scritta sul tasto in bianco, o come una cifra, se è un tasto numerico.

Il modo L (per letters, lettere) si alterna al modo K. In modo L, se non shiftate, il tasto battuto sarà interpretato come il simbolo principale scritto su di esso (nel caso di una lettera, sarà scritta in minuscolo). Sia nel modo K che nel modo L, un tasto premuto insieme a **SYMBOL SHIFT** ritornerà l'elemento scritto in rosso su di esso, un tasto numerico con **CAPS SHIFT** sarà invece interpretato come la funzione di controllo scritta in bianco sopra di esso. Con gli altri tasti **CAPS SHIFT** non ha effetto in modo K, mentre in modo L convertirà i caratteri da minuscolo a maiuscolo.

Il modo C (per capitals, maiuscole) è una variante del modo L nella quale tutte le lettere appariranno in maiuscolo: il **CAPS LOCK** cambia il modo L in modo C e viceversa.

Il modo E (per extended, esteso) è usato per ottenere altri caratteri e, soprattutto, altri comandi; si ottiene dopo aver premuto i due tasti **SHIFT** insieme e dura per una sola battuta. In questo modo, una lettera non shiftata darà il carattere o simbolo scritto in verde sopra essa, una shiftata invece quello che è riportato in rosso sotto di esso. Un tasto numerico darà un simbolo se premuto con **SYMBOL SHIFT** altrimenti esso darà una sequenza di controllo del colore.

Il modo G (per graphics, grafica) si ottiene con **GRAPHICS** (premendo **CAPS SHIFT** ed il **9**), e dura fino a quando lo si preme nuovamente, o fino a quando non premete il **9** da solo. Un tasto numerico produrrà un carattere grafico a mosaico predefinito, e tutte le lettere eccetto v, w, x, y e z daranno un carattere grafico definito dall'utente.

Se desiderate ripetere più volte lo stesso carattere, tenetelo premuto per più di 2 o 3 secondi.

Ciò che scrivete sulla tastiera appare nella parte bassa dello schermo così come è, con i caratteri inseriti subito prima del cursore. Il cursore può essere spostato verso sinistra con il **CAPS SHIFT** ed il **5**, o verso destra con il **CAPS SHIFT** e **8**. Il carattere che precede il cursore può essere cancellato con **DELETE** (**CAPS SHIFT** e **0**), e la riga intera con **EDIT** (**CAPS SHIFT** e **1**) seguito da **ENTER**.

Una linea viene, a seconda, eseguita, introdotta nel programma o usata per leggere dei dati, quando viene premuto **ENTER**. Se c'è un errore non viene accettata e appare un **?** lampeggiante vicino ad esso.

Dopo l'inserimento di ogni linea di programma, compare nella parte alta dello schermo un listato del programma. L'ultima linea introdotta è chiamata linea *corrente* ed è indicata dal simbolo > ; dopo il numero di linea; quest'ultimo può essere spostato usando i tasti (**CAPS SHIFT** e **6**) e (**CAPS SHIFT** e **7**). Se viene premuto il tasto **EDIT** (**CAPS SHIFT** e **1**), la linea corrente è riportata sullo schermo in basso e può essere modificata e/o corretta.

Quando la macchina esegue un comando, o un programma, visualizza i risultati in uscita sulla parte alta dello schermo. Vi rimangono fino a quando viene introdotta una linea di programma, o viene comunque premuto **ENTER**, o usati i tasti o . Nella parte in basso appare un messaggio con un codice (lettera o numero) spiegato nell'appendice B. Rimarrà fino a quando non sarà premuto un qualunque tasto (si è in modo K).

In certi momenti, **CAPS SHIFT** con **SPACE** viene riconosciuto come **BREAK**, e quindi ferma il calcolatore, con un messaggio **D** o **L**; ciò avviene:

(i) al termine di una linea mentre il programma sta girando, o
(ii) mentre il calcolatore sta usando il registratore o la stampante.

Il video

Lo schermo è formato da 24 linee, ognuna lunga 32 caratteri, ed è diviso in due parti. La parte alta è al massimo di 22 linee, e visualizza o un listato, o un risultato in uscita: se uno di questi ultimi è particolarmente lungo, tanto da raggiungere il bordo inferiore, vi sarà lo scorrimento (scrolling), cioè la prima riga sparirà e tutto il listato si sposterà verso l'alto per far spazio all'ultima riga da stampare. Ogni volta che si riempe il video con un ciclo (nel senso di **FOR...NEXT**) il calcolatore si ferma, con il messaggio **scroll**, per evitare che delle righe spariscano prima che possano essere esaminate. Se premete **N**, o **SPACE**, o **STOP** il programma si interromperà e apparirà il messaggio **D BREAK - CONT repeats**; premendo un altro tasto qualsiasi lo scorrimento continuerà. La parte bassa è usata per l'introduzione dei comandi, delle linee di programma e dei dati, oltre che per la visualizzazione dei messaggi. La sezione bassa dello schermo occupa inizialmente solo due linee, ma si espande verso l'alto se è necessario più spazio. Quando, espandendosi, raggiunge la posizione di stampa corrente, ogni ulteriore espansione farà spostare verso l'alto anche la parte superiore. Questo fenomeno si può evidenziare usando due colori diversi per **PAPER** e **BORDER** e leggendo qualche dato molto lungo dopo aver stampato diverse linee.

CAPITOLO
11

Elementi di Programmazione Basic

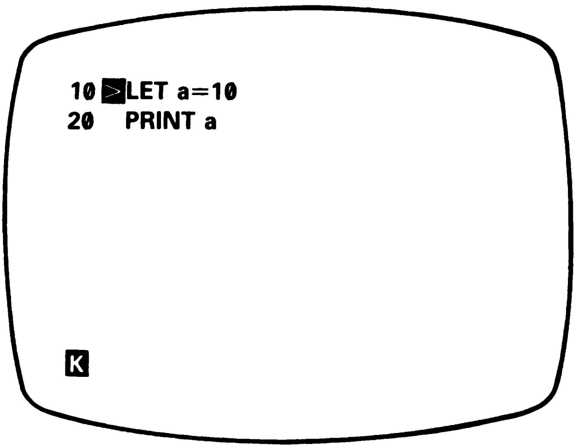
Sommario

Programmi
Numeri di linea
Correzione di programmi usando ▲ , ▶ , ▼ , ◀ e **EDIT**
RUN, LIST
GO TO, CONTINUE, INPUT, NEW, REM, PRINT
STOP durante l'**INPUT** dei dati
BREAK

Scrivete le seguenti due linee di un programma che stampa la somma di due numeri:

```
20 PRINT a
10 LET a=10
```

Lo schermo apparirà come in figura.



Come sapete già, dato che le linee che avete introdotto cominciano con un numero, non sono eseguite immediatamente, ma memorizzate come linee di programma. Dovreste già sapere anche che è il numero di linea a stabilire l'ordine di esecuzione in un programma, che è anche l'ordine in cui le linee vengono listate sul video.

Finora avete una sola variabile: continuate scrivendo

```
15 LET b=15
```

Ovviamente appare tra le altre due linee, cosa che sarebbe stata impossibile se invece di essere numerate 10 e 20 fossero state 1 e 2. Ecco perchè quando si scrive un programma per la prima volta è opportuno usare numeri non contigui per le linee. Si ricordi che i numeri di linea devono essere interi e compresi tra 1 e 9999.

Adesso è necessario modificare la linea 20 in questo modo:

20 PRINT a+b

Potreste facilmente riscrivere la linea per intero, ma molte volte è più comodo usare la funzione **EDIT**. Quando premete il tasto **EDIT**, la linea puntata dal cursore di programma viene riportata per la correzione sulla riga bassa dello schermo. Il cursore di programma, rappresentato da un ▶ tra il numero di linea e la prima parola chiave, indica la linea corrente, di solito l'ultima introdotta, ma può essere spostato con ⇧ o ⇩ .

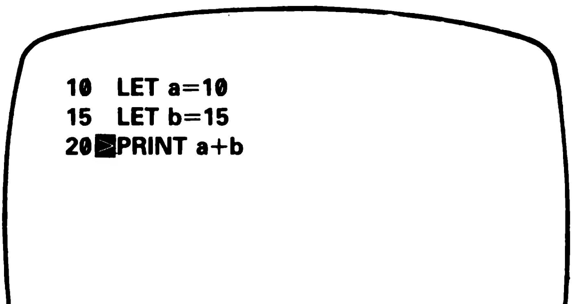
Spostate quindi il ▶ sulla linea 20, premete **EDIT**, quindi muovetevi col tasto fino a portare il cursore alla fine della linea e scrivete

+b (senza **ENTER**)

La linea in basso è ora

20 PRINT a+b

Quando premete **ENTER**, rimpiazzerà la vecchia linea 20, ed il programma apparirà come in figura.



Facendo girare questo programma sarà stampata la somma **a+b**.
Scrivete in modo diretto

PRINT a+b

La somma viene ristampata: infatti le variabili non vengono cancellate quando il programma è terminato.

EDIT può essere agevolmente usato per cancellare tutto quello che si sta scrivendo, e che appare nella parte bassa dello schermo. Scrivete dei caratteri a caso senza **ENTER**: finora l'unico modo che conoscete per liberarvene è di cancellarli uno per volta con **DELETE**. Se premete **EDIT** tutto ciò che avevate introdotto viene rimpiazzato dalla linea corrente. Se ora premete **ENTER**, la linea corrente viene rimessa nel programma lasciandolo inalterato, e la parte bassa dello schermo si libera.

Se introducete una linea per errore, per esempio

12 LET b=8

apparirà nel programma prima che vi accorgiate dell'errore. Per cancellare questa linea, basterà scrivere

12 (con **ENTER**)

che cancellerà la linea 12 (sostituendola idealmente con un vuoto) e farà sparire il cursore di programma. In realtà è nascosto tra la linea 10 e la 15: infatti se premete ⇧ apparirà alla linea 10, se premete ⇩ apparirà alla 15.

Scrivete

12

Il cursore di programma sparirà, ma se premete **EDIT**, sarà la linea 15 ad essere disponibile per la correzione: quando il cursore di programma è nascosto tra due linee, **EDIT** lavora con la successiva.

Scrivete ora

30

Questa volta il cursore di programma è stato nascosto dopo la fine del programma: in questo caso **EDIT** lavora con l'ultima linea, ovvero la 20.

Provate infine

LIST 15

Vedrete apparire sullo schermo

15 LET b=5
20 PRINT a+b

La linea 10 è scomparsa dallo schermo, ma è ancora nel programma: infatti

riappare se premete **ENTER**. L'unico effetto del **LIST 15** è che produce un listato che comincia alla linea 15, e che posiziona il cursore di programma alla linea 15. Quando avrete un programma molto lungo, vi risulterà più comodo usare **LIST**, invece di ⇧ e ⇩ per muovere il cursore.

LIST da solo fa iniziare il listato alla prima linea del programma. Come avete notato, i numeri di linea sono come dei nomi, a cui ci si può riferire un pò come ai nomi delle variabili.

Preparatevi ad introdurre il seguente programma che converte i gradi Fahrenheit in gradi centigradi, pulendo la memoria e le variabili con

NEW

come già visto nel capitolo 5.

```
10 REM conversione temperature
20 PRINT "grad F","grad C"
30 PRINT
40 INPUT "inserire la temperatura in gradi Fahrenheit",F
50 PRINT F,(F-32)*5/9
60 GO TO 40
```

Ovviamente dovrete scrivere per intero il commento alla linea 10, mentre **GO TO**, pur avendo uno spazio in mezzo è un'unica parola chiave (sulla G).

Fate girare il programma, e osservate l'intestazione stampata sullo schermo dalla linea 20. Apparentemente la linea 10 è stata ignorata: effettivamente è solo una linea di commento, che non influenza l'esecuzione del programma. **REM** è l'equivalente di *remark*, che, in inglese, vuol dire nota. Tutto quello che segue la **REM** viene ignorato dal calcolatore.

Si ferma alla **INPUT** della linea 40, aspettando un valore per la variabile **F**. Introducete un numero.

Il calcolatore ha stampato il risultato e sta aspettando un altro numero. Infatti, la linea 60 **GO TO 40**, fa ripetere la linee a partire dalla 40. Introducete un'altra temperatura.

Dopo un pò, quando vorrete finire, invece di introdurre un numero, premete **STOP**. Il calcolatore stampa il messaggio: **D STOP in INPUT in line 30:1**; che vi dice che si è fermato e indica il comando che precede quello che era in esecuzione, il primo e unico della linea 30.

Se volete continuare l'esecuzione da dove l'avevate interrotta, premete

CONTINUE

ed il calcolatore vi chiederà un altro numero.

Con **CONTINUE**, a meno che il messaggio fosse **0 OK**, il calcolatore riprende l'esecuzione col comando successivo a quello specificato nel messaggio, ovvero con quello che stava eseguendo quando è stato interrotto: in questo caso l'**INPUT** alla linea 40.

Il programma funzionerebbe anche se la linea 60 fosse **GO TO 31**. Infatti, se **GO TO** si riferisce ad una linea che non esiste, il programma salta alla linea seguente a quella specificata. **RUN**, seguito da un numero di linea, funziona come **GO TO**. **RUN** da solo equivale a **RUN 0**.

Se fate girare il programma delle temperature fino a riempire lo schermo di numeri, osservate che il calcolatore sposta tutto in su per fare posto all'ultima riga. Questo si chiama scrolling. Dopo aver osservato lo scrolling fermate il programma con **STOP** e ritornate al listato con **ENTER**.

Osservate che nella **PRINT** della linea 50, i due numeri da stampare sono separati da una virgola. La punteggiatura nei programmi è molto importante, e dovrete sempre prestarvi la massima attenzione.

La virgola si usa per far iniziare la stampa sempre allineata su delle colonne prefissate del video. Il punto e virgola fa stampare una cosa attaccata all'altra: provate a sostituire la virgola della linea 50.

Un altro segno di punteggiatura che si può usare nella **PRINT** è l'apostrofo ('): fa ricominciare la stampa all'inizio della linea successiva, come se gli elementi separati dall'apostrofo fossero messi in **PRINT** successive. Infatti, normalmente, la **PRINT** comincia a stampare all'inizio di una riga e la finisce con un ritorno a capo.

Per far sì che la **PRINT** non chiuda la riga con un ritorno a capo, e quindi metta in grado la **PRINT** successiva di continuare a scrivere sulla stessa riga, bisogna scrivere una virgola o un punto e virgola dopo l'ultimo elemento da stampare. Per esempio, sostituite la linea 50 con

```
50 PRINT f,
```

o con

```
50 PRINT f;
```

o con

```
50 PRINT f'
```

La virgola divide tutto in due colonne, il punto e virgola unisce tutto insieme, e l'apostrofo produce una linea bianca per conto suo ma inibisce quella automatica della **PRINT**.

Fate sempre molta attenzione alle virgole ed ai punti e virgola nelle **PRINT**, e soprattutto non confondeteli coi due punti (:) che servono a separare i comandi nella stessa linea.

Aggiungete queste linee:

```
100 REM questo programma ricorda il vostro nome
110 INPUT n$
120 PRINT "Salve"; n$;"!"
130 GO TO 110
```

Costituiscono un programma separato dall'altro. Ma potete memorizzarli insieme nel calcolatore; per farlo girare scrivete

RUN 100

Per ricordarvi che è richiesta una stringa invece di un numero, il calcolatore stampa due virgolette. Eviterete così di scrivere dati sbagliati. Fate girare il programma con i vostri soprannomi preferiti!

Potete anche non usare le virgolette: cancellatele con ⧫ e **DELETE** insieme e scrivete

n$

Visto che mancano le virgolette, il calcolatore capirà di dover fare alcuni calcoli, in questo caso di trovare il valore della stringa alfanumerica **n$**, che corrisponde al nome che le avete assegnato nell'ultimo **RUN**. Naturalmente la linea di **INPUT** agisce come **LET n$=n$** e quindi il valore di **n$** resterà immutato.

La volta dopo, scrivete per verifica

n$

questa volta senza cancellare le virgolette: adesso la vostra stringa avrà il valore "**n$**".

Se volete usare **STOP** durante l'input di una stringa, dovrete riportare il cursore all'inizio della linea, usando ⧫ .

L'istruzione **RUN 100**, usata per far girare il programma, potrebbe sembrare analoga a **GO TO 100**, dato che a prima vista ne svolge le stesse funzioni. In realtà non è così, perchè, a differenza della seconda, la prima azzera tutte le variabili del programma e pulisce lo schermo. Comunque in questo caso particolare, l'uso di **GO TO 100** non avrebbe causato errori durante il programma. Potrete trovarvi in condizioni di voler far girare un programma senza azzerare le variabili; allora ricordatevi di usare **GO TO**. Siate sicuri di avere capito bene le diverse istruzioni.

Inoltre, **RUN** senza alcun numero di linea, fa girare il programma dall'inizio, mentre **GO TO** deve essere necessariamente seguita da un numero.

Avete visto come fermare i programmi scrivendo **STOP** durante l'inserimento di variabili, ma a volte è necessario fermare un programma anche se non state leggendo delle variabili. Per esempio, se avete scritto per errore una linea di questo tipo:

200 GO TO 200

Dando poi

RUN

siete caduti in un ciclo infinito. Premendo **CAPS SHIFT** insieme a **SPACE** (il tasto che ha scritto sopra **BREAK**) il programma si interromperà, e sullo schermo apparirà il messaggio **L BREAK into program**.

Al termine di ogni linea, il programma controlla se questi tasti sono premuti; se effettivamente lo sono, questo si interrompe. **BREAK** può essere usato anche durante il funzionamento del registratore, della stampante e di qualunque altra periferica quando il calcolatore sta aspettando che quest'ultime svolgano un'operazione che non hanno ancora iniziato.

In quest'ultimo caso vi sarà un messaggio differente, **D BREAK-CONT repeats**. In questo, come nella maggior parte dei casi, **CONTINUE** ripete la linea dove il programma si è fermato; ma dopo il messaggio **L BREAK into program, CONTINUE** prosegue con l'esecuzione della linea successiva, senza possibilità di errore.

Adesso fate girare di nuovo il programma del nome, ed inserite:

n$ (dopo aver cancellato le virgolette)

Avete appena scritto una variabile indefinita, causando un errore segnalato col messaggio **2 Variable not found**.

Se adesso scrivete

LET n$= "qualcosa di definito"

(al quale seguirà il messaggio **0 OK, 0:1)** e poi

CONTINUE

troverete che potrete usare il dato **n$** senza problemi.

In questo caso, **CONTINUE**, salta alla linea 110 (di entrata). Esso non considera il riporto dalla linea del **LET** in quanto il messaggio di quest'ultima è "ok" e salta al comando riferito al messaggio precedente (il primo della linea 110), considerato valido. Se un programma si ferma per un errore, potete cercare di correggerlo, e la **CONTINUE** lo farà proseguire.

Come è stato già detto il messaggio **L BREAK into program** è particolare, dato che, dopo di esso, la **CONTINUE** non farà ripetere l'istruzione dove il programma si è fermato.

Potreste essere stati confusi dai listati automatici (cioè quelli che non sono causati da un ordine **LIST** ma seguono l'introduzione di una nuova linea). Per fare delle prove, scrivete un programma composto da 50 linee di commento,

```
1 REM
2 REM
3 REM
:   :
:   :
49 REM
50 REM
```

La prima cosa da ricordare è che la linea corrente apparirà sempre sul video, solitamente in posizione centrale.

Scrivete

LIST

e, dopo aver riempito lo schermo, quando chiede **scroll?** rispondete con **n** (= NO). Apparirà il messaggio **D BREAK-CONT repeats**, come se aveste premuto **BREAK**. Provate anche a premere **y** (= YES, cioè SI) invece di **n**, ed osservate cosa succede. Sperimentate anche che **n, SPACE** e **STOP** contano come NO, mentre qualunque altro tasto conta come SI.

Premete **ENTER** per ritornare al listato delle prime 22 linee e scrivete:

23 REM

e osservate che il listato è dalla linea 2 alla 23. Scrivete ancora

28 REM

e osservate che in quest'altro caso il listato è dalla linea 27 alla 48. Infatti, in ogni caso, avete mosso il cursore di programma producendo un nuovo listato.

Il calcolatore lista il programma in due modi diversi, a seconda della posizione della nuova linea corrente. Se questa precede la prima linea sul video, semplicemente rilista il programma a partire dalla linea corrente. Diversamente lista una linea alla volta, di seguito a quelle già sul video, eseguendo lo scrolling se necessario, fino a quando viene listata la linea corrente. Prima però calcola approssimativamente quanto tempo ci vorrebbe, e se è troppo, comincia a listare, da una linea prossima alla linea corrente, invece che da quella seguente l'ultima sul video.

Provate a muovere la linea corrente, scrivendo

numero di linea **REM**

Il calcolatore ricorda quale è la prima linea mostrata sul video e, ogni volta che si preme **ENTER**, rilista il programma a partire da questa linea. Il comando **LIST** seguito da un numero di linea, sposta la linea corrente alla linea specificata, ma non cambia la linea che il calcolatore ricorda come prima; quindi premendo **ENTER** dopo un **LIST** il calcolatore inizia a listare dalla linea che ricorda come prima, fino alla linea corrente, comunque con le stesse regole viste sopra, ragionando indipendentemente da quello mostrato sul video. Ricordatevi che **LIST** equivale a **LIST 0**: scrivete

LIST

rispondete **y** alla domanda **scroll?** fino a ottenere il listato delle ultime **REM** del programma. Se ora premete **ENTER**, riapparirà il listato delle linee 1-22. Scrivete

LIST 22

e otterrete il listato delle linee 22-43; se premete ancora **ENTER** riotterrete le linee 1-22. Provate

LIST 23

e avrete le linee 23-44. Se premete ancora **ENTER** (dopo aver esaurito le domande **scroll**) otterrete le linee 2-23: la linea corrente 23, deve essere mostrata sul video. Premete ancora

LIST

e rispondete **n** alla domanda **scroll?**. Quindi premete

CONTINUE (abbreviato **CONT**)

Non avete fatto niente dato che la parte bassa dello schermo diventa bianca e non succede niente. Per ritornare alla condizione normale premete **BREAK**. La spiegazione di questo fatto è che la linea di comando in esecuzione conteneva **LIST** come primo comando, cosi chè **CONTINUE** fa ripetere il primo comando, solo che adesso è **CONTINUE** il primo comando che continua a ripetersi fin quando non interviene l'operatore.

Si può ottenere un comportamento diverso sostituendo **LIST** con

:LIST

dopo il quale **CONTINUE**, dà il messaggio **0 OK**. Infatti **LIST** viene considerato il secondo comando della linea, a causa di come è stato scritto, quindi **CONTINUE** fa ripetere il secondo comando. Ma poichè adesso **CONTINUE** è il primo comando, e come secondo comando c'è implicitamente la chiusura della linea, il calcolatore, riprendendo l'esecuzione dal secondo comando, incontra la chiusura e si ferma. Si noti che per linea si intende la linea comandi in modo immediato che cambia ogni volta che si scrive qualcosa. Analogamente provate

::LIST

dopo il quale **CONTINUE** dà **N Statement lost**, infatti salta dopo la chiusura della linea dove non c'è altro.

Finora avete visto i comandi **PRINT, LET, INPUT, RUN, LIST, GO TO, CONTINUE, NEW** e **REM**, e come possano essere usati sia in modo diretto che in un programma, come la maggior parte dei comandi dello ZX Spectrum BASIC. **RUN, LIST, CONTINUE** e **NEW**, non sono comunque molto utili in un programma.

Esercizi

1. Mettete un **LIST** in un programma in modo che si listi da solo.

2. Scrivete un programma che legga dei prezzi da tastiera e stampi il prezzo ivato. Usate una **PRINT** in modo tale che il calcolatore stampi cosa sta facendo e chieda i prezzi in modo molto educato. Modificate il programma per poter introdurre anche l'aliquota IVA.

3. Scrivete un programma per stampare ogni volta che introducete un numero il totale di tutti i numeri introdotti. Suggerimento: usate una variabile **totale** inizializzata a 0 ed una variabile **numero**. Leggete **numero**, sommatelo a **totale**, stampate entrambi e ripetete.

4. Che effetto hanno **CONTINUE** e **NEW** in un programma? Potete immaginarne qualche impiego?

CAPITOLO 12

Decisioni

Sommario
IF, STOP
=, <, >, <=, >=, < >

Finora i programmi esaminati sono stati piuttosto consequenziali, nel senso che le istruzioni venivano svolte senza interruzioni dalla prima all'ultima; questo modo di procedere non sfrutta però una delle funzioni fondamentali del calcolatore: cioè il potere decisionale. Esiste un'importantissima istruzione per il confronto del tipo ...**IF** (se) qualcosa è vero (o falso) **THEN** (allora, in conseguenza) fai qualcosa d'altro.

Usate la **NEW** per cancellare dal calcolatore i precedenti programmi scrivete e fate girare questo programma esempio. (Si tratta chiaramente di un gioco per due persone).

```
10 REM Indovina il numero
20 INPUT a: CLS
30 INPUT "Indovina il numero", b
40 IF b=a THEN PRINT "Hai indovinato": STOP
50 IF b < a THEN PRINT "È troppo piccolo, ritenta"
60 IF b > a THEN PRINT "È troppo grande, ritenta"
70 GO TO 30
```

Avrete visto che l'istruzione **IF** assume la forma

IF condizione **THEN** '...'

dove '...' indica una o più istruzioni, separate dai due punti, come già visto. La condizione è invece un qualcosa che modifica la sequenza normale del programma a seconda che sia vera o falsa: se è vera, cioè se è verificata, il calcolatore svolgerà le istruzioni seguenti il **THEN** se è falsa queste ultime non saranno eseguite e proseguirà nel normale svolgimento del programma dalla linea seguente.

Le condizioni più elementari consistono nel confronto di due numeri o di due stringhe. È possibile controllare se due numeri sono uguali, oppure se uno di questi è maggiore o minore dell'altro; inoltre è possibile controllare se due stringhe sono uguali, oppure se una viene dopo un'altra secondo l'ordine alfabetico. Le relazioni usate sono =, <, >, <=, >= e < >.

= significa “uguale” anche nella **LET** si usa questo simbolo, che però assume un significato completamente differente.

< (**SYMBOL SHIFT** con **R**) significa “è minore di” cosicchè

1<2
−2<−1
−3<1

sono tutte condizioni vere, ma

1<0
0<−2

sono false.

La linea 40 del programma confronta **a** con **b**: se questi sono uguali, il programma è interrotto dalla **STOP**. Il messaggio in basso sullo schermo, cioè **9 STOP, statement, 40:3**, indica che l’istruzione che ha causato l’interruzione del programma è la terza della linea 40.

La linea 50 determina se **b** è minore di **a**, e la linea 60 l’opposto, cioè se **b** è maggiore di **a**. Una di queste condizioni è vera, quindi sarà stampato per forza il commento appropriato, e il programma sarà rieseguito grazie alla linea 70.

L’istruzione **CLS**, cioè “pulisci lo schermo”, della linea 20, serve a non far vedere alle altre persone il numero che avete introdotto.

Inoltre > (**SYMBOL SHIFT** con **T**), significa “è più grande di”, ed è simile a <, ma nell’altro verso. Potete facilmente distinguere questi due simboli, dato che la loro punta è sempre rivolta verso il numero che dovrebbe essere più piccolo.

<= (**SYMBOL SHIFT** con **Q**, da non scrivere come < seguito da =) significa “è minore o uguale di”, e quindi è simile al <, solo che in questo caso la condizione è verificata anche quando i numeri considerati sono uguali: ad esempio 2<=2 è vera, mentre 2<2 no.

>= (**SYMBOL SHIFT** con **E**) significa “è maggiore o uguale di” ed è simile a >, con la differenza sopra spiegata.

<> (**SYMBOL SHIFT** con **W**) significa “non è uguale a”, cioè l’opposto di =.

I matematici scrivono di solito ⩽, ⩾ e # invece di <=, >=, <>. Inoltre essi scrivono relazioni del tipo “2<3<4”, per dire “2<3” e “3<4”, relazioni non possibili in BASIC.

Nota: in alcune versioni di BASIC - ma non dello ZX Spectrum - l’istruzione **IF** può essere espressa nella forma

IF condizione THEN numero di linea

Che equivale a

IF condizione **THEN GO TO** numero di linea

Esercizi

1. Provate a far girare questo programma:

10 PRINT "x": STOP: PRINT "y"

Apparirà sullo schermo la **x**, e si fermerà riportando il messaggio **9 STOP statement, 10:2**. Scrivete ora

CONTINUE

Potreste aspettarvi che salti al comando dove c'è lo **STOP**, dato che **CONTINUE** di solito ripete il comando indicato nel messaggio. In questo caso, sarebbe completamente inutile, dato che il calcolatore si fermerebbe ancora prima di stampare la **y**. **CONTINUE**, però, fa sì che il programma continui con l'istruzione seguente uno **STOP**; quindi, nell'esempio dato, dopo l'introduzione della **CONTINUE** il calcolatore stamperà **y** e il programma terminerà.

CAPITOLO 13

Iterazioni

Sommario
FOR, NEXT
TO, STEP

Con quello che avete imparato finora, se voleste leggere 5 numeri e stamparne la somma, potreste scrivere un programma così (è inutile introdurlo):

```
 10 LET totale=0
 20 INPUT a
 30 LET totale=totale+a
 40 INPUT a
 50 LET totale=totale+a
 60 INPUT a
 70 LET totale=totale+a
 80 INPUT a
 90 LET totale=totale+a
100 INPUT a
110 LET totale=totale+a
120 PRINT totale
```

Ci si rende subito conto che non è un sistema valido per programmare, dato che può andare bene per 5 numeri, per 10, ma non certo per 100 numeri.

È molto meglio creare una variabile di controllo che conti fino a 5 e poi fermi il programma, come in quest'altro esempio (introducetelo):

```
10 LET totale=0
20 LET giro=1
30 INPUT a
40 REM giro=numero di INPUT eseguite
50 LET totale=totale+a
60 LET giro=giro+1
70 IF giro<=5 THEN GO TO 30
80 PRINT totale
```

Notate che sarebbe molto facile cambiare la linea 70 per leggere 10, 100, 1000 numeri.

Questa iterazione è così utile che esistono due apposite istruzioni per renderla più facilmente implementabile: sono la **FOR** e la **NEXT** che si usano sempre in coppia.

Ecco come si usano nel programma esempio:

```
10 LET totale=0
20 FOR i=1 TO 5
30 INPUT a
40 REM i=numero di INPUT eseguiti
50 LET totale=totale+a
60 NEXT i
80 PRINT totale
```

(Non è necessario riscrivere completamente il programma, dovete semplicemente correggere le linee 20, 40 e 70 dell'esempio precedente).

Notate che la variabile **i** svolge la stessa funzione di **giro**, infatti è la variabile di controllo del ciclo **FOR-NEXT**. Il nome di questa deve sempre essere formata da una sola lettera, di solito si usa **i, j, k, n** o **m**.

In questo programma le linee 30, 40, 50, sono eseguite per i 5 valori di **i**; 1, il valore di inizializzazione, 2, 3, 4, e 5, il valore limite. Quindi, quando **i** ha coperto i suoi 5 valori, viene eseguita la linea 80.

Si può anche fare in modo che la variabile di controllo venga incrementata ad ogni giro di un valore a piacere invece che di 1. Per fare ciò si usa **STEP** seguito dall'incremento, collocato subito dopo il valore limite. La sintassi per il comando generalizzato è

FOR variabile di controllo = valore iniziale **TO** limite **STEP** incremento

dove la variabile di controllo è una singola lettera, il valore iniziale, il limite e l'incremento sono espressioni calcolabili dalla macchina come numeri: per esempio numeri in cifre, somme, nomi di variabili numeriche, ecc. Se sostituite la linea 20 del programma con

```
20 FOR i=1 TO 5 STEP 3/2
```

la **i** coprirà i valori 1, 2.5 e 4. Notate che si possono usare anche numeri decimali e che la variabile di controllo non deve necessariamente coincidere con il limite. Il ciclo si ripete fino a quando la variabile di controllo è minore o uguale al limite. Prendete bene nota di questo: quando la variabile è uguale al limite, il ciclo viene eseguito per l'ultima volta e la variabile incrementata ancora; quindi la variabile esce dal ciclo con un valore *sempre superiore al limite.*

Provate questo programma che stampa i numeri da 1 a 10 al contrario:

```
10 FOR n=10 TO 1 STEP -1
20 PRINT n
30 NEXT n
```

Ovviamente, con incrementi negativi, il limite si trova ad essere minore della

variabile di controllo, quindi non si potrebbe avere alcun ciclo se il controllo fosse eseguito nel solito modo. Con incrementi negativi il ciclo viene ripetuto fino a quando la variabile di controllo è minore o uguale al limite.

Più cicli **FOR-NEXT** possono essere eseguiti uno dentro l'altro, bisogna però fare attenzione che non si incrocino, vale a dire che il ciclo più interno sia chiuso prima della ripetizione del ciclo immediatamente più esterno. Esaminate l'esempio che stampa le combinazioni dei pezzi del domino:

```
10 FOR m=0 TO 6
20 FOR n=0 TO m
30 PRINT m; ":"; n; " ";
40 NEXT n
50 PRINT
60 NEXT m
```

Osservate che il ciclo **n** è completamente interno al ciclo **m**; i due cicli sono impiegati correttamente. Quello che bisogna evitare è di avere due cicli **FOR-NEXT** che si sovrappongano senza che nessuno dei due sia completamente interno all'altro, come in questo programma sbagliato:

```
5 REM programma sbagliato
10 FOR m=0 TO 6
20 FOR n=0 TO m
30 PRINT m; ":"; n; " ";
40 NEXT m
50 PRINT
60 NEXT n
```

Due cicli **FOR-NEXT** devono essere o uno interamente interno all'altro, o totalmente separati.

Bisogna anche fare attenzione a non saltare dall'interno di un ciclo **FOR-NEXT** ad una linea di programma fuori dal ciclo. Infatti, saltando il **NEXT** che chiude il ciclo (quando la variabile ha raggiunto il limite), il calcolatore lascia in sospeso la variabile di controllo inizializzata dalla **FOR** e rischia di confondersi. In certi casi si avrà il messaggio di errore **NEXT without FOR** oppure **variable not found**.

FOR e **NEXT** possono anche essere usati in un comando diretto. Provate per esempio:

```
FOR m=0 TO 10: PRINT m: NEXT m
```

Lo stesso comando può essere usato in modo un pò artificioso per ripetere indefinitamente un'istruzione in modo immediato, cosa impossibile altrimenti dato che in modo immediato non ci sono numeri di linea e quindi **GO TO** non si può usare. Infatti:

```
FOR m=0 TO 1 STEP 0: INPUT a: PRINT a: NEXT m
```

si ripete indefinitamente, dato che, con incremento 0, la variabile di controllo non raggiunge mai il valore limite.

Non si raccomanda comunque di seguire questa procedura, poichè in caso di errore, il comando va perso e **CONTINUE** non funziona.

Esercizi

1. Una variabile di controllo non ha solo un nome ed un valore, come le variabili normali, ma anche un limite, un incremento e un puntatore che punta al comando seguente la **FOR**. Convincetevi che tutte queste informazioni sono create quando viene eseguita la **FOR** e che servono anche alla **NEXT**, per sapere di quanto incrementare la variabile di controllo, se saltare indietro e dove.

2. Fate girare il terzo programma di questo capitolo e verificate il valore della variabile **c**. Perchè ha valore 6 e non 5? Cosa succede se aggiungete **STEP 2** nella linea 20?

3. Modificate il terzo programma in modo che invece di sommare 5 numeri ne sommi una quantità a vostro piacere. Cosa succede se facendo girare il programma dite che volete sommare 0 numeri? Perchè vi potreste aspettare che questo causi dei problemi al calcolatore anche se è chiaro cosa significa? (Il calcolatore cerca il comando **NEXT c**, in questo caso non necessario). Considerate attentamente il problema.

4. Alla linea 10 del quarto programma cambiate **10** in **100** e fatelo girare; stamperà i numeri da 100 a 89 e chiederà **scroll?** per darvi la possibilità di vedere i numeri prima che vengano cancellati dal video. Verificate che se premete **n, STOP** o **BREAK** il programma si ferma stampando il messaggio **D BREAK-CONT repeats**, e che se premete qualunque altro tasto stampa altre 22 linee.

5. Cancellate la linea 30 dal quarto programma e fatelo girare. Stamperà il primo numero e si fermerà col messaggio **0 OK**. Verificate che se scrivete

```
NEXT n
```

il programma ripeterà il giro stampando il prossimo numero.

CAPITOLO 14

Subroutines

Sommario
GO SUB, RETURN

A volte capita che differenti parti di un programma debbano svolgere compiti uguali o molto simili, e che quindi si riscrivano le stesse linee due o più volte. È più conveniente, specie se le linee da scrivere sono numerose, metterle in un certo punto del programma e richiamarle ogni volta. Una tale struttura di programma è chiamata subroutine ed è gestita in BASIC da due apposite istruzioni, la **GO SUB** e la **RETURN**. Una subroutine è un insieme di linee di programma la cui esecuzione inizia da una linea data e la cui ultima istruzione, in sequenza logica, è **RETURN**. La sintassi di **GO SUB** è:

GO SUB n

dove n è il numero di linea della prima linea della subroutine. Funziona a tutti gli effetti come **GO TO** n, ma il calcolatore ricorda dov'era la **GO SUB** per poter riprendere l'esecuzione del programma dopo aver incontrato **RETURN**. In pratica, il calcolatore mette il numero di linea e il numero di comando nella linea in cima allo stack della **GO SUB**.

RETURN

toglie l'ultimo indirizzo dallo stack ed esegue il comando immediatamente successivo.

Per esempio, riprendete in considerazione il programma per indovinare un numero, e riscrivetelo come segue:

```
 10 REM "Gioco indovina il numero modificato"
 20 INPUT a: CLS
 30 INPUT "Indovina il numero",b
 40 IF a=b THEN PRINT "Esatto": STOP
 50 IF a<b THEN GO SUB 100
 60 IF a>b THEN GO SUB 100
 70 GO TO 30
100 PRINT "riprova"
110 RETURN
```

Il comando **GO TO** alla linea 70 è molto importante, perchè altrimenti il

programma proseguirebbe nella subroutine causando l'errore **7 RETURN without GO SUB** alla linea 110. Fateci sempre molta attenzione, perchè è un errore ricorrente.

Ecco un altro semplice programma che illustra l'uso di **GO SUB**:

```
100 LET x=10
110 GO SUB 500
120 PRINT s
130 LET x=x+4
140 GO SUB 500
150 PRINT s
160 LET x=x+2
170 GO SUB 500
180 PRINT s
190 STOP
500 LET s=0
510 FOR y=1 to x
520 LET s=s+y
530 NEXT y
540 RETURN
```

Fate girare questo programma e cercate di capire cosa fa. La subroutine inizia alla linea 500.

Una subroutine ne può comodamente chiamare un'altra, e può anche chiamare sè stessa. In questo caso si dice *recursiva*. Quando scrivete programmi non abbiate paura di creare molte subroutines concatenate; generalmente è una buona pratica di programmazione.

CAPITOLO 15

READ, DATA, RESTORE

Sommario
READ, DATA, RESTORE

Nei programmi precedenti abbiamo visto come le informazioni o i dati possano essere introdotti da tastiera ad ogni esecuzione del programma usando la **INPUT**, cosa che può diventare molto noiosa se ogni volta bisogna introdurre gli stessi dati, specie se i dati sono molti. È utile, in questi casi, memorizzare i dati permanentemente all'interno del programma con l'istruzione **DATA** e leggerli con **READ** e **RESTORE**. Per esempio:

```
10 READ a,b,c
20 PRINT a,b,c
30 DATA 10,10,30
40 STOP
```

Un comando **READ** è formato dall'istruzione **READ** seguita da una lista di nomi di variabili, separati da virgole. Funziona in modo simile alla **INPUT**, ma legge i dati contenuti nella **DATA** invece di quelli introdotti dalla tastiera.

Ogni comando **DATA** è una lista di espressioni, numeriche o stringhe (le stringhe devono essere racchiuse tra apici), separate da virgole. Le **DATA** possono essere messe in qualunque punto del programma, dato che il calcolatore le ignora fino a quando non esegue una **READ**. Tutti i dati specificati nelle varie **DATA** sparse nel programma formano idealmente un'unica lista di dati, da cui il calcolatore preleva in ordine i dati richiesti dalla **READ**. C'è un puntatore che indica l'ultimo dato letto, cosicchè ogni **READ** legge il dato seguente all'ultimo letto, a partire dal primo nel programma. Se si eseguono più **READ** dei dati disponibili, senza eseguire una **RESTORE**, il calcolatore dà errore.

Notate che non serve usare la **DATA** in modo immediato, perchè la **READ** non riuscirebbe a trovare i dati.

Osservate come queste lavorino nel programma esempio. La linea 10 dice al calcolatore di leggere tre dati ed assegnarli alle variabili **a, b** e **c**. La linea 20 le fa stampare. La linea 30 contiene i valori per **a, b** e **c**. La linea 40 chiude il programma.

Le informazioni nella **DATA** possono anche essere messe all'interno di un ciclo **FOR-NEXT**; scrivete:

```
10 FOR n=1 TO 6
20 READ D
30 DATA 2,4,6,8,10,12
40 PRINT D
50 NEXT n
60 STOP
```

Quando il programma viene fatto girare, potete osservare la **READ** leggere tutti i valori della **DATA**. Provate anche con variabili stringa, come nell'esempio:

```
10 READ d$
20 PRINT "Oggi è il",d$
30 DATA "1 marzo 1983"
40 STOP
```

Finora avete visto il modo più lineare di leggere le espressioni dalla lista **DATA**, ovvero iniziando col primo dato e andando avanti fino alla fine. Comunque è possibile spostare il puntatore della lista **DATA**, tramite il comando **RESTORE**. **RESTORE** deve essere seguito da un numero di linea e sposta il puntatore al primo elemento della prima **DATA** a partire dalla linea specificata. Come al solito, se si omette il numero di linea, viene assunto uguale a 0. Per concludere, provate il seguente programma:

```
10 READ a,b
20 PRINT a,b
30 RESTORE 60
40 READ x,y,z
50 PRINT x,y,z
60 DATA 1,2,3
70 STOP
```

In questo programma alla linea 10 **a** assume il valore 1, **b** il valore 2. La **RESTORE 10** resetta il puntatore, cosicchè la linea 50 rilegge i dati a partire dal primo. Se non siete sicuri di aver capito fate girare questo programma senza la linea 30.

CAPITOLO 16

Espressioni

Sommario
Operazioni: +, −, *, /
Espressioni, notazione scientifica, nomi di variabili

Avete già visto come lo ZX Spectrum può eseguire dei conti usando le 4 operazioni aritmetiche, +, −, *, / (ricordatevi che l'asterisco indica la moltiplicazione, la barra la divisione) e come può trovare il valore di una variabile dandone il nome.

L'esempio

LET ivato=somma*15/100

è solo una banale dimostrazione di come queste operazioni possano essere combinate. Una combinazione di operazioni, ad esempio **somma*15/100**, si chiama espressione quindi un'espressione è semplicemente un modo di dire in breve al calcolatore di eseguire diverse operazioni, in una certa sequenza. In effetti, l'espressione **somma*15/100** significa: prendi il valore della variabile **somma**, moltiplicalo per 15 e poi dividilo per 100.

Prima di proseguire, a meno che non l'abbiate già fatto, leggete i primi capitoli introduttivi, dove viene spiegato come lo ZX Spectrum gestisce i numeri e l'ordine in cui esegue le operazioni.

È opportuno ricapitolare questi concetti: le moltiplicazioni e le divisioni sono eseguite per prime, cioè hanno una priorità più alta rispetto alle addizioni e sottrazioni. Relativamente una all'altra, le moltiplicazioni e le divisioni hanno la stessa priorità: ciò significa che sono eseguite in ordine da sinistra verso destra; quando sono state esaurite, vengono calcolate le addizioni e le sottrazioni, che, analogamente a quanto detto sopra, hanno priorità identica fra loro e quindi saranno svolte da sinistra verso destra, fino a esaurimento. Per quanto voi non abbiate bisogno di alcun artificio per tenere a mente l'ordine di svolgimento delle operazioni in una data espressione, il calcolatore lo fa assegnando ad ogni operazione un numero di priorità compreso tra 1 e 16. Il * ed il / hanno una priorità 8, + e − hanno priorità 6: ovviamente più alto è il numero, più alta è la priorità.

L'ordine dei calcoli è assolutamente rigoroso, ma può essere modificato usando le parentesi; tutto quello contenuto tra le parentesi viene calcolato prima, e poi trattato come numero singolo. Le espressioni sono di uso comune e molto utili perchè, quando il calcolatore richiede o vuole usare un numero, gli si può fornire un'espressione, e automaticamente userà come numero il valore dell'espressione in quel momento, tenendo conto delle eventuali variabili. È quasi sempre possibile, quando non lo è, viene indicato esplicitamente. Anche con le stringhe si possono

formare delle espressioni; esse possono essere sommate a piacimento, usando anche le parentesi. Dovreste già sapere quali sono i nomi validi che si possono usare per le variabili. Riassumendo, un nome di variabile-stringa deve essere seguito dal $, ed il nome di una variabile di controllo per un ciclo **FOR-NEXT** deve essere una lettera singola; i nomi delle variabili numeriche ordinarie sono più liberi possono contenere qualunque lettera o cifra, ma il loro primo carattere deve essere una lettera. Possono essere introdotti anche gli spazi nei nomi delle variabili, ma vengono ignorati; vale a dire che se due variabili differiscono solamente per alcuni spazi introdotti qua e là fra i vari caratteri, saranno considerate dal sistema come un'unica variabile. Anche le maiuscole e le minuscole sono considerate uguali.

Ecco qualche esempio di nomi di variabili valide:

x
p42
questo nome è così lungo che io non sarò mai in grado di riscriverlo tutto senza fare un errore
ora siamo in sei (gli ultimi due nomi sono considerati identici dal calco-
oRASiamoInSeI latore; hanno quindi lo stesso valore)

Esempi di nomi di variabili non ammessi:

2001 (comincia con una cifra)
3 orsi (comincia con una cifra)
M*A*S*H (* non è una lettera o una cifra)
Fotherington-Thomas (– non è una lettera o una cifra)

Le espressioni numeriche possono essere rappresentate da un numero elevato ad un esponente; riguardo a questo argomento consultate i capitoli introduttivi. Provate ora le seguenti **PRINT**:

```
PRINT 2.34e0
PRINT 2.34e1
PRINT 2.34e2
```

e così via fino a

```
PRINT 2.34e15
```

Come avete notato, il calcolatore, dopo un pò, inizia ad usare la notazione scientifica infatti non può usare più di 14 caratteri consecutivi per scrivere un numero. Analogamente provate:

PRINT 2.34e−1
PRINT 2.34e−2

e così via.

PRINT stampa solamente 8 cifre significative di un numero provate:

PRINT 4294967295,4294967295−429e7

Rendetevi conto che il calcolatore memorizza tutte le cifre di 4294967295, anche se non è in grado di mostrarle tutte insieme, cioè di stamparle.

Lo ZX Spectrum usa l'aritmetica in virgola mobile; ovvero memorizza in modo separato le cifre di un numero (la sua mantissa), e la posizione del punto decimale (l'esponente). Questa notazione presenta numerosi vantaggi, ed è impiegata dalla stragrande maggioranza dei calcolatori in commercio, di qualsiasi dimensione essi siano. Tuttavia non conserva i numeri con precisione assoluta, come ovvio. La precisione dei numeri nello ZX Spectrum è di circa 9,5 cifre significative: ovviamente anche i numeri interi sono rappresentati con questa precisione. Per esempio, provate:

PRINT 1e10+1−1e10,1e10−1e10+1

Avrete la stampa di **0** e **1** infatti nel primo caso il calcolatore non riesce a distinguere tra **1e10**+1 e **1e10**, cosicchè il risultato della sottrazione è 0 nel secondo caso sottrae lo stesso numero a sè stesso e poi aggiunge 1, dando quindi come risultato 1. In questo esempio la rappresentazione usata dal calcolatore fornisce il numero reale con un errore di approssimazione di circa 2, e poichè 1 è più piccolo di quest'errore, il calcolatore non può assolutamente distinguere tra i due numeri. Provate un altro esempio, ancora più particolare:

PRINT 5e9+1−5e9

L'imprecisione della rappresentazione di 5e9 è solo di circa 1, e in effetti l'uno che viene sommato sarà arrotondato per eccesso a 2; è come se avessimo in realtà i due numeri 5e9+2, e 5e9+1, che al calcolatore appaiono uguali. Questa precisione sarà comunque sufficiente per tutte le applicazioni.

Il numero intero più grande che può essere memorizzato con assoluta precisione è (2 elevato a 32) − 1, uguale a 4294967295.

La stringa "" senza caratteri, è chiamata stringa *vuota* o *nulla*. Ricordatevi che gli spazi sono importanti in una stringa, e che una stringa formata da spazi non è la stessa cosa di una stringa vuota. Provate:

PRINT "Hai finito di leggere "I Promessi Sposi" Luigi"?

Quando premete **ENTER**, vedrete il punto interrogativo lampeggiante indicarvi un errore da qualche parte nella linea. Quando il calcolatore trova le virgolette all'inizio di **I Promessi Sposi** immagina che indichino la fine della frase **"Hai finito di leggere"**, e quindi non riesce a capire il significato della frase **I Promessi Sposi**.

Se dovete stampare le virgolette in una **PRINT**, l'unico modo per ottenerle consiste nell'introdurle due volte, come in questo esempio:

PRINT "Hai finito di leggere ""I Promessi Sposi"" Luigi?"

Come potete vedere da quello stampato sullo schermo, ogni coppia di virgolette viene stampata solamente una volta quindi avete dovuto scriverle due volte solo per permettere al calcolatore di capire che le volevate stampate, e che non indicavano la fine della stringa.

CAPITOLO

17

Stringhe

Sommario
Divisione di stringhe (slicing) usando **TO** (notazione non standard in BASIC).

Data una stringa più lunga di un carattere, è sempre possibile estrarre da essa delle sottostringhe formate da caratteri consecutivi della stringa data. Estendendo questo concetto, "stringa" è una sottostringa di "stringa lunga", ma "l stringa" e "gala ring" non lo sono.

Il BASIC dello ZX Spectrum accetta una notazione non standard per definire una qualunque sottostringa a partire da una qualunque stringa, fornendogli il carattere di partenza e quello finale. L'espressione è:

stringa (primo carattere **TO** ultimo carattere)

dove per primo ed ultimo carattere si intendono le posizioni dei caratteri della stringa principale con cui si vuole che la sottostringa inizi e termini. Così per esempio:

"abcdef"(2 TO 5)="bcde"

Se omettete il carattere di partenza, la sottostringa inizierà dal primo carattere della stringa origine, se omettete il carattere finale, la sottostringa comprenderà tutti i caratteri a partire da quello indicato fino all'ultimo carattere della stringa origine; per esempio:

"abcdef"(TO 5)="abcdef" (1 TO 5)="abcde"
"abcdef"(2 TO)="abcdef"(2 TO 6)="bcdef"
"abcdef"(TO)="abcdef"(1 TO 6)="abcdef"

(Nell'ultimo esempio si sarebbe potuto mettere anche **"abcdef"()**).

Se si desidera ottenere una sottostringa formata dal solo carattere ennesimo della stringa origine, basterà racchiudere il numero del carattere voluto tra parentesi; così

"abcdef"(3)="abcdef"(3 TO 3)="c"

Come regola generale, sia il carattere iniziale che quello finale devono esistere nella stringa origine; tuttavia, se il carattere finale è specificato come carattere

precedente al carattere di destinazione, la stringa verrà considerata come nulla e la macchina non darà errore. Per esempio:

"abcdef"(5 TO 7)

dà come errore **3 subscript wrong**, perchè dato che la stringa contiene solo 6 caratteri, non potrà essere considerato un settimo; inoltre

"abcdef"(8 TO 7)=""

o

"abcdef"(1 TO 0)=""

I numeri che identificano i caratteri iniziali e finali non devono assolutamente essere negativi, altrimenti si avrà l'errore **B integer out of range**. Il programma seguente esemplifica meglio queste regole:

```
10 LET a$="abcdef"
20 FOR n=1 TO 6
30 PRINT a$(n TO 6)
40 NEXT n
50 STOP
```

Dopo aver provato e capito questo programma, date il comando **NEW** e scrivete:

```
10 LET a$="COME STAI?"
20 FOR n=1 TO 10
30 PRINT a$(n TO 10),a$((10-n) TO 10)
40 NEXT n
50 STOP
```

È anche possibile assegnare un segmento di una stringa data invece di estrarlo. Provate per esempio:

LET a$="Io sono lo ZX Spectrum"

e poi

LET a$(4 TO 7)="****"**

e

PRINT a$

Notate che la sottostringa **a$**(4 TO 7) è composta da 4 caratteri, e quindi sono stati usati solo i primi 4 asterischi. Quando si assegna una sottostringa non è necessario preoccuparsi del numero di caratteri che si cerca di assegnare; il calcolatore provvede automaticamente a troncare a destra l'eccedenza, oppure ad aggiungere spazi per raggiungere la lunghezza richiesta.

Provate adesso:

LET a$()="Ciao"

e poi

PRINT a$;"."

Notate che è avvenuta l'inserzione di spazi, dato che **a$**() conta come sottostringa lunga quanto tutta la stringa **a$**.

LET a$="Ciao"

funzionerà in modo normale riducendo la lunghezza di **a$**.

Espressioni più complesse potranno aver bisogno di parentesi per poter essere interpretate correttamente. Considerate:

"abc"+"def"(1 TO 2)="abcde"
("abc"+"def")(1 TO 2)="ab"

Esercizi

1. Provate a scrivere un programma per stampare il giorno della settimana, usando la divisione delle stringhe. Partite da una stringa unica che contenga le abbreviazioni di tutti i giorni della settimana, cioe "LunMarMerGioVenSabDom".

CAPITOLO 18

Funzioni

Sommario

DEF
LEN, STR$, VAL, VAL$, SGN, ABS, INT, SQR
FN

Pensate alla macchina per fare il ragù. Mettete un pezzo di carne da una parte, girate una manovella, ed esce il ragù dall'altra parte. Con carne di maiale fate ragù di maiale, col pesce fate il ragù di pesce e con carne di manzo fate il ragù di manzo.

Le funzioni sono molto simili ad una macchina per fare il ragù, ma lavorano con numeri e stringhe invece che con la carne. Voi introducete nella funzione il valore iniziale (l'*argomento*), lo elaborate facendo alcuni calcoli, ed ottenete un altro numero, il risultato.

Carne nella → Macchina per il ragù → Esce il ragù
Argomento nella funzione → Funzione → Uscita del risultato

Argomenti diversi danno risultati diversi e, se l'argomento non è valido, la funzione non sarà eseguita e darà errore. Non tutte le funzioni sono uguali. Così come esistono macchine per fare il ragù, macchine per lavare i piatti, ecc., ci sono funzioni per eseguire tutti i tipi di calcoli. Diverse funzioni produrranno risultati diversi dagli stessi argomenti.

Le funzioni possono essere usate nelle espressioni, scrivendo il nome della funzione seguito dall'argomento. Quando l'espressione viene calcolata, il risultato della funzione è sostituito al posto della funzione e usato per calcolare il valore dell'espressione.

Per esempio, esiste una funzione chiamata **LEN**, che calcola la lunghezza di una stringa. Il suo argomento è la stringa di cui si vuole conoscere la lunghezza, il risultato è la lunghezza. Provate a scrivere

PRINT LEN "Sinclair"

il calcolatore scriverà la risposta 8, cioè il numero delle lettere della parola "Sinclair". Per ottenere la funzione **LEN**, come la maggior parte delle funzioni, dovete usare il modo esteso: premete **CAPS SHIFT** e **SYMBOL SHIFT** contemporaneamente per far apparire la **E**, nel cursore, e poi usate il tasto **L**.

Se nella stessa espressione usate insieme funzioni ed operazioni, le funzioni sono calcolate prima delle operazioni. Ancora una volta, comunque, l'ordine naturale dei calcoli può essere alterato dalle parentesi. Per esempio, ecco due espressioni che differiscono soltanto per le parentesi. I calcoli sono eseguiti in modo totalmente diverso, per quanto il risultato finale sia lo stesso.

```
LEN "Mario"+ LEN "Rossi"    LEN ("Mario"+"Rossi")
5+LEN "Rossi"               LEN ("MarioRossi)
 5+5                        LEN "MarioRossi"
  10                        10
```

STR$ converte i numeri in stringhe: il suo argomento è un numero, e il risultato è la stringa che apparirebbe sullo schermo se il numero fosse usato con una **PRINT**. Ha quindi come risultato una stringa formata da un carattere per ogni cifra del numero. Notate come il suo nome finisca con un dollaro. Il dollaro dopo un nome di funzione indica che il risultato è una stringa. Per esempio, potete scrivere

LET a$=STR$ 1e2

che dà esattamente lo stesso risultato di

LET a$="100"

Oppure potete scrivere

PRINT LEN STR$ 100.0000

e ottenere 3 come risposta, perchè **STR$** di 100.0000 è uguale a "100".

VAL è il contrario di **STR$**: converte stringhe in numeri. Per esempio,

VAL "3.5"=3.5

In effetti, se prendete un numero come argomento di **STR$**, e il risultato di questa come argomento di **VAL**, riottenete il numero di partenza.

Attenzione però, che se fate il contrario, cioè prendete una stringa come argomento di **VAL**, e il risultato come argomento di **STR$**, non sempre riotterrete la stringa di partenza: cercate di capire perchè.

VAL è una funzione molto potente, dato che la stringa che abbiamo usata come argomento può essere una qualunque espressione numerica. Quindi, per esempio

VAL "2*3"= 6

o perfino

VAL ("2"+"*3")= 6

In questo caso l'elaboratore procede su due livelli, prima l'argomento di **VAL** è calcolato come stringa. L'espressione stringa **"2"+"*3"** è calcolata per dare la stringa **"2*3"** quindi la stringa viene privata delle virgolette, e quello che rimane è calcolato come numero, così **2*3** è calcolato per dare **6**.

È facile che vi confondiate se non vi prestate attenzione. Cercate di capire la ragione di ogni virgoletta

PRINT VAL “VAL“““VAL““““““2””””””””””””

(Ricordate che dentro una stringa, per ottenere le virgolette, è necessario scriverle due volte. Proseguendo in concatenamenti di stringhe, vi accorgerete che le virgolette devono essere quadruplicate o perfino ottuplicate).

Esiste un'altra funzione piuttosto simile alla **VAL**, anche se è meno utile, la **VAL$**. Il suo argomento è ancora una stringa, ma anche il risultato è una stringa, differentemente dalla **VAL**. Per vedere come funziona, ricordatevi che **VAL** funziona in due livelli: prima l'argomento è valutato come stringa, e poi la stringa viene calcolata come un numero. Con **VAL$** il primo livello è uguale, ma dopo, qualunque cosa sia rimasta, viene calcolata come un'altra stringa, quindi

VAL$ “““““Succo di frutta”””””=“Succo di frutta”

(Notate come le virgolette si moltiplichino ancora). Scrivete

LET a$=“99”

e provate ad eseguire le seguenti funzioni: **VAL a$, VAL “a$”, VAL “““““a$”””””**, **VAL$ a$, val$ “a$”** e **VAL$ “““““a$”””””**. Alcune di queste funzionano, altre no: cercate di spiegare il perchè delle risposte. Prendetevela con calma!

SGN è la funzione di *segno*. È la prima funzione che avete incontrato che non ha niente a che fare con le stringhe, infatti sia il suo argomento che il suo risultato sono numeri. Il risultato è +1 se l'argomento è positivo, 0 se l'argomento è 0 e −1 se l'argomento è negativo.

ABS è un'altra funzione numerica. Converte un qualunque numero in un numero positivo, eliminandone il segno. Quindi, per esempio

ABS −3.2 = ABS 3.2 = 3.2

INT sta per “parte intera di” dove l'argomento e il risultato sono numeri anche negativi. Questa funzione converte un numero decimale in uno intero, ignorando la parte decimale; quindi, per esempio

INT 3.9 = 3

Fate attenzione quando applicate questa funzione ai numeri negativi, perchè arrotonda sempre al numero negativo più basso. Per esempio

INT −3.9 = −4

SQR estrae la radice quadrata di un numero: ciò significa che il risultato moltiplicato per sè stesso ridà l'argomento. Per esempio

SQR 4 = **2** perchè 2*2=4
SQR 0.25 = **0.5** perchè 0.5*0.5=0.25
SQR 2 = **1.4142136** (valore approssimato)
perchè 1.4142136*1.4142136=2.0000001

Se moltiplicate un qualunque numero, anche negativo, per sè stesso, la risposta è sempre positiva: questo significa che i numeri negativi non hanno radici quadrate, così se applicate **SQR** ad un numero negativo, otterrete il messaggio d'errore **An Invalid Argument**.

Voi potete anche definire le vostre funzioni personali. I nomi che potete usare per le vostre funzioni devono avere la forma **FN** seguito da una lettera, se il risultato è un numero, **FN** seguito da una lettera ed un dollaro se il risultato è una stringa. Con le vostre funzioni personali l'argomento deve *essere sempre racchiuso tra parentesi.*

Per definire una funzione, usate il comando **DEF** in una qualunque parte del programma. Ecco per esempio la definizione della funzione **FN s** il cui risultato è il quadrato dell'argomento:

10 DEF FN s(x)=x*x: REM il quadrato di x

DEF si ottiene nel modo esteso, usando **SYMBOL SHIFT** e **1**. Quando usate la **DEF**, il calcolatore automaticamente scriverà **FN** dopo la **DEF**; infatti **DEF** è usata solo seguita da **FN**. Dopo questo, **s** completa il nome **FN s** della funzione.

La **x** tra parentesi è il nome a cui vi volete riferire come argomento della funzione. Può essere una qualunque lettera singola, o se l'argomento è una stringa, una qualunque lettera seguita da dollaro.

Dopo il segno uguale, viene la vera definizione della funzione, cioè l'espressione che viene calcolata ogniqualvolta usiate la funzione. In questa espressione, quando usate la lettera che avete usato come argomento (in questo caso la **x**), viene vista come una variabile ordinaria.

Quando avete definito una funzione, potete usarla analogamente alle funzioni già definite dal calcolatore, scrivendo il suo nome (**FN s**), seguito dall'argomento. Ricordate che quando vi riferite a funzioni definite da voi, l'argomento deve essere racchiuso tra parentesi. Provate qualche volta, dopo aver introdotto la linea di definizione riportata più sopra:

PRINT FN s(2)
PRINT FN s(3+4)
PRINT 1+INT FN s (LEN "gallo"/2+3)

In alcuni dialetti BASIC, diversamente che nello ZX Spectrum, è necessario racchiudere tra parentesi anche l'argomento delle funzioni residenti nel calcolatore.

Avete visto che **INT** arrotonda per difetto. Per arrotondare all'intero più vicino, potete definire una funzione che sommi .5 prima di eseguire la **INT**:

```
20 DEF FN r(x)=INT (x+0.5): REM dà una x arrotondata al più vicino intero.
```

e otterrete come risultato, per esempio

```
FN r(2.9) = 3          FN r(2.4) = 2
FN r(−2.9) = −3        FN r(−2.4) = −2
```

Confrontate i risultati che avete ottenuto con quelli che avreste ottenuto utilizzando **INT** invece di **FN r**.

Inserite e fate girare il seguente programma:

```
10 LET x=0: LET y=0: LET a=10
20 DEF FN p(x,y)=a+x*y
30 DEF FN q()=a+x*y
40 PRINT FN p(2,3),FN q()
```

Ci sono diverse cose da notare in questo programma. Primo, una funzione non deve avere per forza un solo argomento: ne può avere più di uno, o addirittura nessuno (ma anche in questo caso è necessario usare la parentesi).

Secondo, non importa in quale parte del programma mettiate il comando **DEF**. Nell'esempio, dopo aver compiuto ed eseguito la linea 10, salta le linee 20 e 30, ed esegue la linea 40. Il comando **DEF** deve essere comunque in qualche parte del programma, non può essere in un comando diretto.

Terzo, **x** e **y** sono sia nomi di variabili del programma, sia nomi di argomenti nella funzione **FN p**. In questo caso succede che, quando il calcolatore sta eseguendo una funzione, i valori passati agli argomenti hanno il sopravvento sui valori originali delle variabili, e quindi per la funzione **FN p**, **x** e **y** hanno il valore assegnatoli alla chiamata. Subito dopo l'esecuzione della funzione, **x** e **y** tornano ad avere i valori originali. Così, quando è calcolata **FN p(2.3)**, **a** ha valore 10, perchè, non essendo stata usata come argomento, mantiene il valore iniziale, **x** ha il valore 2 perchè è il primo argomento, **y** ha il valore 3 perchè è il secondo argomento, e quindi il risultato è 10+2*3=16.Invece, quando **FN q()** è calcolata, non ci sono argomenti, così **a**, **x** e **y** si riferiscono tutte alle variabili originali, e valgono 10, 0 e 0 rispettivamente: la risposta in questo caso è 10+0*0=10.

Modificate la linea 20 in

```
20 DEF FN p(x,y)=FN q()
```

Questa volta, **FN p(2,3)**, darà come risultato 10, perchè **FN q** userà i valori originali di **x** e **y** invece degli argomenti di **FN p**.

Alcuni BASIC (non quello dello ZX Spectrum) hanno funzioni chiamate

LEFT$, RIGHT$, MID$ e TL$.

LEFT$ (a$,n) ritorna una sottostringa di a$ formata dai suoi primi n caratteri.

RIGHT$ (a$,n) ritorna una sottostringa di a$ formata dai caratteri dall'ennesimo alla fine.

MID$ (a$,n1,n2) ritorna una sottostringa di a$ formata da n2 caratteri partendo dall'n1esimo.

TL$ (a$) ritorna una sottostringa di a$ formata da tutti i suoi caratteri eccetto il primo.

Sullo ZX Spectrum tutte queste funzioni sono rimpiazzate dalla funzione **TO**. Provate a ricrearle come funzioni definite dall'utente, per esempio:

```
10 DEF FN t$(a$)=a$(2 TO): REM TL$
20 DEF FN l$(a$,n)=a$(TO n): REM LEFT$
```

Verificate che queste lavorino anche con lunghezze di stringa di 0 e 1.

Notate che la **FN l$** ha due argomenti, uno numerico e l'altro stringa. Una funzione può avere fino a 26 argomenti numerici e, analogamente, fino a 26 argomenti stringa (26 come le lettere dell'alfabeto inglese).

Esercizi

Usate la funzione **FN s(x)=x*x** per provare la **SQR**: troverete che

FN s(SQR x)=x

funziona per un qualunque **x** positivo, e

SQR FN s(x)=ABS x

funziona sia che **x** sia positivo che negativo (perchè **ABS?**).

CAPITOLO 19

Funzioni matematiche

Sommario

↑
PI, EXP, LN, SIN, COS, TAN, ASN, ACS, ATN

Questo capitolo parla delle funzioni matematiche trattate dallo ZX Spectrum. Se non avrete mai bisogno di usarne alcuna e troviate faticoso studiarle, non abbiate paura di saltarle. Saranno trattate le operazioni di elevamento a potenza, la funzione **EXP** e **LN**, le funzioni trigonometriche seno, coseno, tangente (**SIN, COS, TAN**) ed i loro inversi (**ASN, ACS, ATN**). Elevare a potenza un numero significa moltiplicare il numero per sè stesso tante volte quanto è la potenza. In matematica questo si indica scrivendo la potenza in piccolo in alto a destra di ciò che si vuole elevare; purtroppo il calcolatore usa un'altra notazione. Per elevare a potenza, scrivete il numero che dovete elevare, la base, seguito dalla ↑ e seguito quindi dall'esponente. Per esempio, le potenze di 2 sono:

2↑1=2	
2↑2=2*2=4	(2 al quadrato, scritta normalmente 2^2)
2↑3=2*2*2=8	(2 al cubo, scritta normalmente 2^3)
2↑4=2*2*2*2=16	(2 alla quarta, scritta normalmente 2^4)

Nel caso più semplice, questo significa a↑b, cioè a moltiplicato per sè stesso b volte, il che ha senso ovviamente solo se b è un numero positivo intero. Per trovare una definizione che lavori anche per altri valori di b, ricordiamo la regola

a↑(b+c)=a↑b*a↑c

(Notate che l'elevamento a potenza ha una priorità più elevata delle moltiplicazioni e delle divisioni, così in un'espressione composta gli elevamenti a potenza sono eseguiti per primi e dopo i * e i /). Non dovreste aver bisogno di ulteriori spiegazioni quando b e c sono entrambi numeri positivi interi; ma se avete bisogno di elevare a potenze numeri che non lo siano, allora veramente potreste avere dei dubbi:

a↑0=1
a↑(−b)=1/a↑b
a↑(1/b)= la b esima radice di a, cioè il numero che, moltiplicato per sè stesso b volte, dà a)
e
a↑(b*c)=(a↑b)↑c

Se queste cose sono nuove per voi, non cercate di ricordarvele immediatamente,

ma limitatevi a sapere che

a↑(−1)=1/a
e
a↑(1/2)=**SQR** a

e può darsi che, quando queste vi saranno più familiari, il resto inizierà ad avere significato. Provate tutto ciò con il seguente programma

```
10 INPUT a,b,c
20 PRINT a↑(b+c)=a↑b*a↑c
30 GO TO 10
```

Naturalmente, la prima regola indicata è ancora vera, quindi ogni volta che la linea 20 viene eseguita, il calcolatore stampa due numeri uguali. Notate che, a causa del modo in cui lavora il calcolatore, la base, la **a** in questo caso, non può mai essere negativa.

Un esempio piuttosto tipico sull'uso di queste funzioni è quello dell'interesse composto. Supponete che teniate del denaro in una banca, e che vi diano il 15% di interesse all'anno; quindi alla fine dell'anno, non solo avrete il 100% di quello che avete depositato, ma anche il 15% di interesse che la società vi ha dato. Avrete quindi il 115% di quello che avevate originariamente depositato. Per esprimersi diversamente, avete moltiplicato il vostro denaro per 1.15, e ciò ovviamente è vero qualunque cifra abbiate depositato. Dopo un altro anno, accadrà la stessa cosa, ma con la cifra del primo anno aumentata del 15%: quindi avrete 1.15*1.15=1.15↑2=1,3225 volte la somma originale. In generale, dopo y anni, avrete 1.15 elevato alla y per la cifra che avete depositato.

Provate questo comando:

```
FOR y=0 TO 100: PRINT n,10*1.15↑y: NEXT y
```

e che, anche se iniziate con 10 lire, il capitale aumenta abbastanza rapidamente, e, cosa più importante, aumenta tanto più velocemente mentre si va avanti, per quanto, come potete verificare, non riesce a stare al passo con l'inflazione.

Questo tipo di comportamento, cioè quando dopo un certo intervallo di tempo, una quantità si moltiplica per una proporzione fissa, è chiamata crescita esponenziale, ed è calcolata elevando un numero prefissato alla potenza del tempo. Supponete di aver definito la seguente funzione:

```
10 DEF FN a(x)=a↑x
```

Qui **a** dovrebbe essere stato assegnato in una **LET**; il suo valore corrisponde al tasso di interesse, che varia solo di tanto in tanto.

Esiste un certo valore di **a** che fa sembrare questa funzione particolarmente interessante per l'occhio esperto di un matematico: questo valore è chiamato *e*. Lo ZX Spectrum ha una funzione chiamata **EXP** definita come

EXP x = e↑x

Sfortunatamente, *e* da solo non è un numero particolarmente carino, dato che è un numero decimale infinito non periodico. Potrete vedere i suoi primi decimali scrivendo

PRINT EXP 1

dato che **EXP 1 = e↑1 = e**. Naturalmente questa è solo un'approssimazione, non potrete mai scrivere *e* esattamente.

LN

L'inverso della funzione esponenziale è la funzione logaritmica: il logaritmo in base a di un numero x è la potenza alla quale dovete elevare a per avere il numero x, e si scrive $log_a x$. Quindi, per definizione, a↑$log_a x$=x; e quindi anche log(a↑x)=x.

Può darsi che abbiate studiato l'uso dei logaritmi in base 10, logaritmi decimali, o volgari, per fare le moltiplicazioni. Lo ZX Spectrum ha una funzione **LN**, che calcola i logaritmi naturali di base *e*. Per calcolare i logaritmi in una qualunque altra base, dovete dividere il logaritmo naturale per il logaritmo della base che desiderate avere:

$log_a x$=**LN x/LN a**

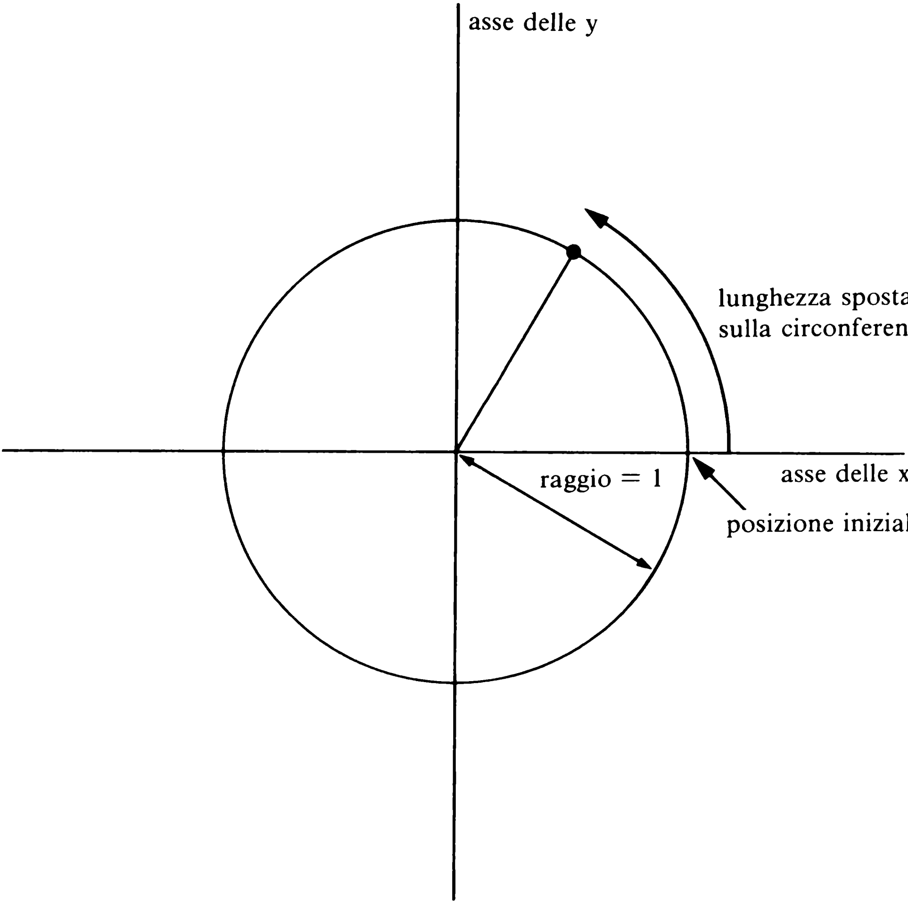
PI

Come probabilmente sapete, dato un cerchio, si può ricavare il suo perimetro (la lunghezza della circonferenza) moltiplicando il diametro per un numero chiamato pi greco (π è la lettera greca p, ed è usata come abbreviazione di perimetro).

π è, come *e*, un numero decimale di infinite cifre non periodiche: le prime cifre sono 3.141592653589... Sullo Spectrum esiste una funzione **PI** che ritorna questo numer provate **PRINT PI**.

SIN, COS e TAN; ASN, ACS e ATN

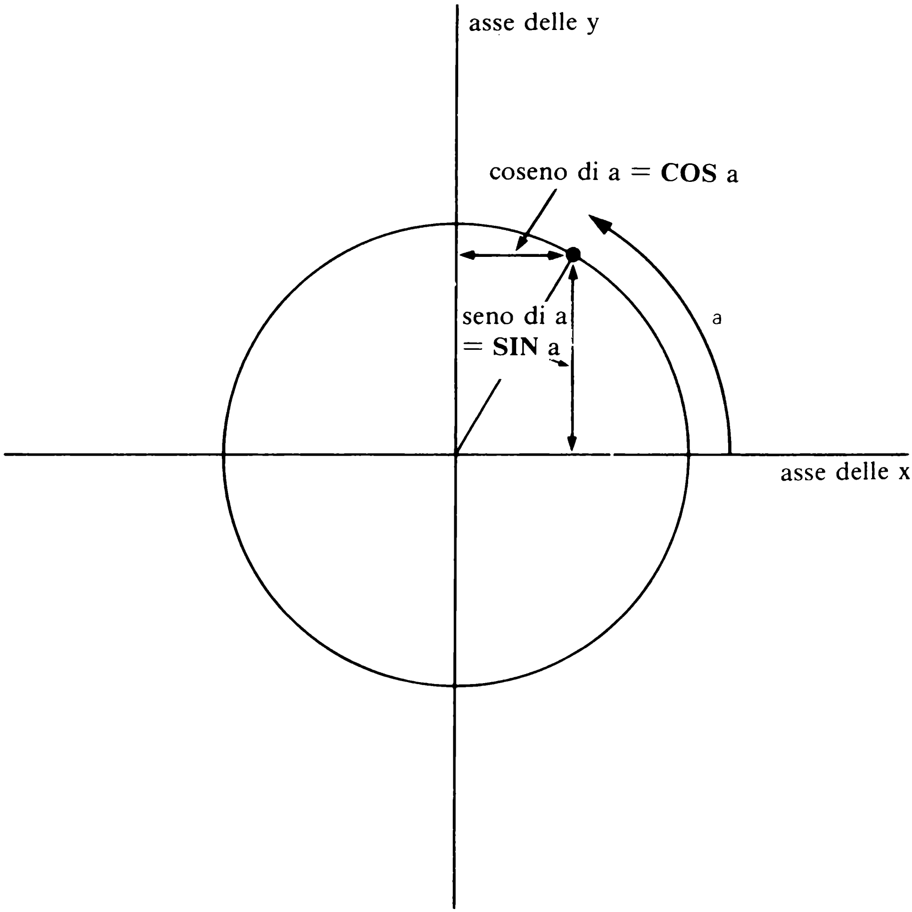
Le funzioni trigonometriche permettono di sapere cosa succede quando un punto si muove su una circonferenza. Supponete di avere un cerchio di raggio 1 (assumete cioè il raggio del cerchio come unità di misura), e supponete di avere un punto che si muova sulla circonferenza a partire dalla posizione delle tre del quadrante dell'orologio e che si muova in senso antiorario.



Disegnate anche gli assi passanti per il centro del cerchio. L'asse che va dalle 3 alle 9 dell'orologio è chiamato asse delle x, e l'asse che va dalle 12 alle 6 è chiamata asse delle y.

Per specificare dove si trova il punto sulla circonferenza, sarà sufficiente sapere di quanto si è spostato dalla posizione delle 3; chiamiamo questa distanza sulla circonferenza a. Sappiamo che la circonferenza è lunga 2π, dato che se il raggio è 1, il diametro è 2: così, quando il punto si sarà spostato di un quarto di cerchio, a sarà uguale a n/2, quando si sarà spostato di mezzo cerchio, a sarà uguale a π, e, al giro completo, a sarà uguale a 2π.

Una volta che sapete così dove si trova il punto sulla circonferenza, potreste aver bisogno di sapere quanto a destra dell'asse delle y o quanto di sopra all'asse delle x si trovi. Queste misure sono rispettivamente il seno ed il coseno di a. Le funzioni **COS** e **SIN** del calcolatore ritornano queste misure.



Notate che se il punto si sposta alla sinistra dell'asse delle y il coseno diventa negativo, e che se il punto si sposta sotto l'asse delle x il seno diventa a sua volta negativo.

Notate anche che, una volta che il punto ha compiuto un giro completo, cioè che ha superato 2π, è tornato al punto di partenza, ed il seno ed il coseno ricominciano ad avere gli stessi valori infatti:

SIN (a+2*PI) = SIN a
COS (a+2*PI) = COS a

La tangente di a è semplicemente il seno diviso il coseno; la corrispondente funzione sul calcolatore è chiamata **TAN**.

A volte sarà necessario l'uso delle funzioni inverse, cioè quelle funzioni che

danno il valore di a dati il seno, il coseno o la tangente. Queste funzioni sono l'arcoseno, l'arcocoseno e l'arcotangente (rispettivamente sul calcolatore **ASN, ACS** e **ATN**).

Osservando il diagramma del punto intorno al cerchio, notate che misurare a non è altro che un modo di sapere l'angolo formato dal raggio che unisce il centro del cerchio col punto, con l'asse delle x. Quando $a=\pi/2$, l'angolo è di 90 gradi, per $a=\pi$ l'angolo è uguale a 180 gradi e così via, fino a quando a sarà pari a 2π e l'angolo sarà di 360 gradi. A volte è più comodo dimenticarsi dei gradi, e misurare l'angolo in ragione di a solo. Si dice allora che stiamo misurando l'angolo in radianti. Quindi, $\pi/2$ radianti = 90 gradi e così via.

Tenete sempre ben presente che sullo ZX Spectrum e nella maggior parte dei BASIC, **SIN, COS**, ecc. usano sempre e solo radianti. Per convertire i gradi in radianti dividete per 180 e moltiplicate per π. Per convertire da radianti in gradi, dividete per π e moltiplicate per 180.

CAPITOLO 20

Numeri a caso

Sommario
RANDOMIZE
RND

Questo capitolo illustra la funzione **RND** ed il comando **RANDOMIZE**. Entrambi hanno a che fare con i numeri a caso, e dovrete fare attenzione a non confonderli: sono entrambi sullo stesso tasto, la **T**, dove **RANDOMIZE** è stato abbreviato in **RAND**.

RND è una funzione, fa dei calcoli, e produce un risultato, solo che non ha bisogno di un argomento; ogni volta che la usate, avete come risultato un numero a caso tra 0 e 1. Qualche volta potrebbe uscire 0, ma mai 1. Provate:

```
10 PRINT RND
20 GO TO 10
```

per vedere come varia il numero. Siete in grado di dedurre qualche regola? Non dovreste esserlo, dato che "a caso" significa proprio senza regole.

Comunque **RND** non ritorna numeri perfettamente casuali, perchè segue una sequenza prefissata di 65536 numeri casuali. In ogni caso, questi numeri sono mescolati così bene, che non vi sono apparentemente regole di generazione; diciamo quindi che **RND** è una funzione *pseudo-casuale*.

Come detto, **RND** genera un numero a caso compreso tra 0 e 1: è molto facile modificare quest'intervallo, per poter ottenere un numero in un campo qualsiasi. Per esempio, **5*RND** ritorna un numero a caso compreso tra 0 e 5, e **1.3+0.7*RND** genera numeri tra 1.3 e 2. Per avere numeri a caso interi, usate **INT**, (ricordate che questa funzione arrotonda per difetto) come in **1+INT(RND*6)**, che simula un dado. **RND*6** è nel campo da 0 a 6, ma siccome non raggiungerà mai il 6, **INT(RND*6)** può generare 0,1,2,3,4 e 5: ecco perchè sommiamo 1. Ecco un programma per simulare il lancio di due dadi:

```
10 REM lancio dei dadi
20 CLS
30 FOR n=1 TO 2
40 PRINT 1+INT (RND*6); " ";
50 NEXT n
60 INPUT a$: GO TO 20
```

Premete **ENTER** ogni volta che volete lanciare i dadi.

Il comando **RANDOMIZE n** è usato per far sì che **RND** inizi ad estrarre i numeri da un punto predefinito della sequenza, come potete vedere in questo programma:

```
10 RANDOMIZE 1
20 FOR n=1 TO 5: PRINT RND,: NEXT n
30 PRINT : GO TO 10
```

Dopo ogni esecuzione di **RANDOMIZE 1**, la sequenza di **RND** comincia con il numero 0.0022735596. Con **RANDOMIZE** è possibile usare un qualunque numero compreso tra 1 e 65535 per ottenere diversi punti d'inizio per la sequenza **RND**.

Se un programma che avete scritto contiene degli errori che non avete ancora trovato e usa la funzione **RND**, sarà utile l'uso di **RANDOMIZE**, in modo tale che il programma si comporti esattamente alla stesso modo ogni volta che lo farete girare.

RANDOMIZE da solo è, come al solito, equivalente a **RANDOMIZE 0**, ma ha un effetto diverso da **RANDOMIZE** seguito da un numero. Infatti fa si che la sequenza **RND** cominci in una posizione a caso, come potete osservare nel seguente programma:

```
10 RANDOMIZE
20 PRINT RND : GO TO 10
```

La sequenza che ottenete non è veramente a caso, dato che **RANDOMIZE** usa il tempo trascorso dall'accensione del calcolatore; dato che in questo programma viene incrementato della stessa quantità ogni volta che **RANDOMIZE** viene eseguito, la **RND** seguente ritorna più o meno allo stesso valore. Con questa procedura avreste ottenuto numeri più casuali usando **GO TO 20** anzichè **GO TO 10** alla linea 20.

Sappiate che la maggior parte dei dialetti BASIC usano **RND** e **RANDOMIZE** per generare numeri a caso, ma in modo non proprio uguale. Ecco un programma che simula il lancio di una moneta e conta il numero di teste e di croci:

```
10 LET teste=0: LET croci=0
20 LET moneta=INT (RND*2)
30 IF moneta=0 THEN LET teste=teste+1
40 IF moneta=1 THEN LET croci=croci+1
50 PRINT teste;",";croci,
60 IF croci<>0 THEN PRINT teste/croci;
70 PRINT : GO TO 20
```

Il rapporto tra le teste e le croci dovrebbe diventare circa 1 se fate molti lanci, perchè la probabilità di avere o testa o croce è uguale.

Esercizi

1. Verificate che, se scegliete un numero **n** compreso tra 0 e 872 e scrivete **RANDOMIZE n**, il valore del primo **RND** che fate è

(75*(il vostro numero+1)−1)/65536

2. (Per soli matematici).
Sia p un numero primo grande, e sia a una radice primitiva modulo p.
Quindi, se b_i è il residuo iesimo di a^i modulo p ($1 \leqslant b_i \leqslant p-1$), la sequenza

$$\frac{bi-1}{p-1}$$

è una sequenza ciclica di p-1 numeri distinti nell'intervallo 0,1 (escludendo 1). Scegliendo una a appropriata, questi numeri possono simulare una sequenza di numeri a caso.

65537 è un numero primo di Fermat, della forma $2^{16}+1$. Dato che il gruppo moltiplicativo di residui non nulli modulo 65537, ha la potenza di 2 come suo ordine, un residuo è una radice primitiva se e soltanto se non è un residuo quadratico. Applicando il teorema di Gauss sui reciproci quadratici si dimostra che 75 è una radice primitiva modulo 65537.

Lo ZX Spectrum usa p=65537, a=75 e memorizza alcuni b_i-1. **RND** funziona sostituendo b_i-1 in memoria con $b_{i+1}-1$ e conservando il risultato ($b_{i+1}-1$, (p−1). **RANDOMIZE** n (con $1 \leqslant n \leqslant 65535$) rende b_i uguale a n+1.

RND è distribuito molto uniformemente nell'intervallo 0, 1.

CAPITOLO 21

Matrici

Sommario

Matrici (il modo in cui lo ZX Spectrum lavora con stringhe di matrici non è standard)
DIM...

Supponete di avere una lista di numeri, per esempio i voti di dieci persone in una classe. Per memorizzarli nel calcolatore potreste inizializzare una variabile per ogni persona. Potreste decidere di chiamare la variabile Allievo 1, Allievo 2 e così via fino all'Allievo 10, ma otterreste un programma per leggere i numeri piuttosto noioso da scrivere; sarebbe molto bello poter scrivere un programma come il seguente:

```
 5 REM questo programma non girerà
10 FOR n=1 TO 10
20 READ Allievo n
30 NEXT n
40 DATA 10,2,5,19,16,3,11,1,0,6
```

Purtroppo non si può.

Comunque quest'idea può essere applicata mediante un altro meccanismo, cioè usando le matrici. Una matrice è un insieme di variabili i cui elementi hanno tutti lo stesso nome, e si distinguono solo per un numero (l'indice), scritto tra parentesi dopo il nome. Nel nostro esempio, il nome potrebbe essere *b* (come le variabili di controllo del ciclo **FOR-NEXT**, il nome delle matrici deve essere composto da una singola lettera), e le dieci variabili potrebbero essere *b*(1), *b*(2), e così via fino a *b*(10).

Gli elementi di una matrice si chiamano *variabili con indice*, per distinguerli dalle variabili semplici che già conoscete.

Prima che possiate usare una matrice, è necessario che riserviate dello spazio nella memoria del calcolatore: potete farlo con un comando **DIM** (per dimensionamento).

```
DIM b(10)
```

inizializza una matrice chiamata *b* di dimensione 10, cioè vi sono 10 variabili indicizzate *b*(1)... *b*(10), e inizializza i dieci valori a 0. Inoltre cancella qualunque altra matrice di nome *b* esistente precedentemente. Notate che una matrice e una variabile semplice con lo stesso nome non sono la stessa cosa, e possono quindi essere usate contemporaneamente, badando a non confonderle o scambiarle.

L'indice può essere una qualunque espressione numerica, così è possibile scrivere:

```
10 FOR n=1 TO 10
20 READ b(n)
30 NEXT n
40 DATA 10,2,5,19,16,3,11,1,0,6
```

Potete anche definire matrici a più dimensioni. In una matrice a due dimensioni sono necessari due numeri per specificare un elemento, analogamente alle linee ed alle colonne che indicate per trovare la posizione di un carattere sullo schermo. Infatti una matrice a due dimensioni è del tutto simile ad una tabella. Potete anche immaginarvi le righe e le colonne di una pagina stampata per le due dimensioni, ed il numero di pagina come terza dimensione. Naturalmente stiamo parlando di variabili numeriche, così gli elementi non saranno caratteri, come in un libro, ma numeri. Cercate di pensare agli elementi di una matrice *v* a tre dimensioni, come specificati da *v* (numero di pagine, numero di linee, numero di colonne).

Per esempio, per inizializzare una variabile a due dimensioni di nome **c**, con dimensioni 3 e 6, usate il seguente comando **DIM**

DIM c(3,6)

che vi mette a disposizione 3*6 = 18 variabili indicizzate.

c(1,1), c(1,2), ..., c(1,6)
c(2,1), c(2,2), ..., c(2,6)
c(3,1), c(3,2), ..., c(3,6)

Lo stesso principio si può estendere a matrici di qualunque dimensione.

Per quanto possiate avere numeri e matrici con lo stesso nome, non potete avere due matrici con lo stesso nome, anche se hanno differenti dimensioni. Potete anche dimensionare matrici di stringhe. Le stringhe in una matrice sono differenti dalle semplici stringhe, per il fatto che sono di lunghezza fissa. Quando si assegna una variabile stringa in una matrice, se la stringa che si deve inserire è più corta, si riempiono i caratteri in più con spazi, se è troppo lunga si tagliano i caratteri in eccesso a destra. Potete facilmente pensare alle matrici di stringhe come matrici di caratteri con una dimensione supplementare. Il nome di una matrice di stringhe è una lettera seguita dal dollaro; una matrice di stringhe e una stringa semplice non possono avere lo stesso nome, diversamente dalle variabili numeriche.

Supponete di voler dimensionare una matrice **a$** di 5 stringhe. Innanzitutto dovete decidere la lunghezza massima delle stringhe: per esempio, supponiamo 10 caratteri. Quindi potete scrivere:

DIM a$(5,10) (scrivetelo)

Questa linea dimensiona una matrice di 5*10 caratteri, ma immaginate che ogni riga contenga una stringa, così

a$(1)=a$(1,1) a$(1,2) ... a$(1,10) (prima stringa)
a$(2)=a$(2,1) a$(2,2) ... a$(2,10) (seconda stringa)
. . . .
. . . .
a$(5)=a$(5,1) a$(5,2) ... a$(5,10) (quinta stringa)

Se quando leggete una matrice, specificate lo stesso numero di indici usato in **DIM**, in questo caso due, ottenete un carattere singolo, dato che avete una matrice di caratteri; se trascurate l'ultimo indice, otterrete la stringa di lunghezza fissa. Quindi, per esempio, a$(2,7) è il settimo carattere nella stringa a$(2); è anche possibile usare la notazione (slicing) per suddividere le stringhe scrivendo a$(2)(7). Scrivete adesso:

LET a$(2)="1234567890"

e

PRINT a$(2),a$(2,7)

otterrete

1234567890   7

Invece dell'ultimo indice, quello che potete tralasciare, potete usare un comando per dividere la stringa, quindi, per esempio,

a$(2,4 TO 8)=a$(2)(4 TO 8)="45678"

Ricordatevi:

in una matrice di stringhe, tutte le stringhe hanno la stessa lunghezza prefissata. Il comando **DIM** per le matrici di stringhe ha una dimensione extra che specifica questa lunghezza. Quando estraete una variabile indicizzata da una matrice di stringa, dovete inserire un numero in più o un divisore di stringa che corrisponde al numero extra nel comando **DIM**.

È anche possibile dimensionare matrici di stringhe senza dimensioni scrivete:

DIM a$(10)

e scoprirete che **a$** si comporta come una normalissima variabile stringa, soltanto che ha lunghezza 10 e le assegnazioni seguono le regole delle matrici.

Esercizi

1. Usate i comandi **READ** e **DATA** per riempire una matrice **m$** di dodici stringhe, in cui **m$**(n) è il nome dell'**n**esimo mese dell'anno. (Suggerimento: la linea **DIM** sarà **DIM m$**(12,9). Controllate stampando tutti gli **m$**(n) in un ciclo).
   Scrivete

   PRINT "questo è il mese di";m$(4);"ed è bello andare in giro"

   Cosa potete fare per gli spazi di troppo?

CAPITOLO 22

Condizioni

Sommario
AND, OR
NOT

Nel capitolo 12 abbiamo visto che un comando **IF** si scrive nella forma

IF condizione **THEN** ...

In quel capitolo avevate visto che le condizioni potevano essere le relazioni (=, <, >, <=, >= e <>), che confrontano due numeri o due stringhe. Ora vedrete come è possibile combinare diverse condizioni mediante gli operatori logici **AND, OR** e **NOT**.

L'operatore logico **AND** richiede che la prima relazione e la seconda relazione siano verificate perchè la condizione globale della **IF** sia soddisfatta. Per esempio,

IF a$="sì" AND x > 0 THEN PRINT x

e otterreste la stampa di **x** solo se sia **a$** è uguale a **SI**, sia **x** maggiore di 0. Il BASIC è un linguaggio molto simile all'inglese, in effetti gli operatori logici hanno esattamente la stesso significato che hanno le parole omonime in inglese: **AND** significa "e", **OR** significa "oppure" e **NOT** significa "non". Cercate di pensare all'uso degli operatori logici nella grammatica.

Quando l'operatore logico **OR** unisce due condizioni, è sufficiente che o una o l'altra siano vere oppure entrambe, perchè sia verificata la condizione globale. Notate che in grammatica però, con la "o", non sempre si include il caso in cui entrambe le condizioni siano verificate.

L'operatore logico **NOT** inverte semplicemente la condizione. Si dice anche che questo operatore nega la condizione. Una condizione negata è vera solo se la condizione originaria è falsa, una condizione negata è falsa solo se la condizione originaria è vera, cioè esattamente il contrario. Proprio come il "non" della grammatica.

Le espressioni logiche possono essere costruite usando relazioni matematiche e gli operatori **AND, OR** e **NOT**, esattamente allo stesso modo in cui le operazioni numeriche possono essere fatte coi numeri, ed i +, −, ecc.. Anche nelle operazioni logiche è possibile usare le parentesi, se necessario: anche le operazioni logiche hanno le priorità, analogamente alle operazioni matematiche. L'ordine delle priorità nelle operazioni logiche è, dalla più bassa alla più alta, **OR, AND, NOT**, le relazioni matematiche e le operazioni aritmetiche. **NOT**, in effetti, è una funzione con un argomento e un risultato, ma la sua priorità è molto più bassa delle altre

funzioni, quindi il suo argomento non ha bisogno di essere racchiuso tra parentesi, a meno che non contenga **OR** o **AND**. **NOT a=b** significa lo stesso che **NOT (a=b)** e lo stesso di **a<>b**, naturalmente.

<> è la negazione di uguale, dato che è vero solo se = è falso, in altre parole

a<>b è lo stesso di **NOT a=b**

e anche

NOT a<>b è lo stesso di **a=b**

Convincetevi bene che se >= e <= sono rispettivamente le negazioni di < e >, voi potete sempre eliminare un **NOT** da una relazione, semplicemente cambiando opportunamente quest'ultima.

E anche,

NOT (prima relazione logica, e **AND**, seconda condizione)

è lo stesso che

NOT (prima relazione) **OR NOT** (seconda)

e

NOT (prima relazione, **OR** seconda relazione)

è lo stesso che

NOT (prima) **AND NOT** (seconda).

Ricordatevi bene queste regole. Usandole, voi potete spostare i **NOT** fino ad applicarli direttamente alle relazioni aritmetiche e quindi eliminarli modificando le relazioni stesse. Parlando logicamente, **NOT** non è necessario, anche se scoprirete che può rendere una relazione più chiara.

La seguente parte del capitolo si addentra in dettaglio nelle relazioni logiche, ed è abbastanza complessa: saltatela, se non vi sentite sicuri!

Provate

PRINT 1=2,1<>2

vi aspettereste di avere un errore di sintassi, ma per quanto riguarda il calcolatore, non c'è differenza tra un numero normale ed un valore logico, solo che questi ultimi sono soggetti a qualche regola.

(i) =, <, >, <=, >= e <> danno sempre i seguenti risultati numerici: 1, se la relazione è vera, e 0 se la relazione è falsa. Conseguentemente il comando **PRINT** sopra, stamperà uno 0 per 1=2 (ovviamente falso) ed un 1 per 1<>2 (che è vero).

(ii) In

IF condizione **THEN** ...

la condizione può essere qualunque espressione numerica, se il suo valore è 0, conta come se fosse falso, e qualunque altro valore, includendo ovviamente il valore di 1 (ritornato da una relazione vera), conta come vero. Quindi, un comando **IF** significa esattamente la stessa cosa che

IF condizione <> 0 **THEN** ...

(iii) **AND, OR** e **NOT** sono anche applicabili a numeri:

x **AND** y ha il valore: $\begin{cases} x\text{, se } y \text{ è vera (non 0)} \\ \text{0 (falsa), se } y \text{ è falsa (0)} \end{cases}$

x **OR** y ha il valore: $\begin{cases} 1 \text{ (vero), se } y \text{ è vera (non 0)} \\ x\text{, se } y \text{ è falsa (0)} \end{cases}$

NOT x ha il valore: $\begin{cases} 0 \text{ (falso), se } x \text{ è vero (non 0)} \\ 1 \text{ (vero), se } x \text{ è falso (0)} \end{cases}$

(Notate che "vero" significa "non 0" quando si controlla un valore dato, ma significa "1" quando se ne produce uno nuovo).

Rileggete il capitolo una seconda volta alla luce di quanto avete appena imparato, e verificate che tutto funzioni secondo queste regole.

Nelle espressioni **x AND y, x OR y** e **NOT x, x** e **y** avranno di solito i valori 0 e 1 per falso e vero. Scrivete tutte le differenti combinazioni (sono 10, 4 per la **AND**, 4 per la **OR** e 2 per la **NOT**), e verificate che effettivamente si comportino come avete potuto capire leggendo il capitolo. Provate questo programma:

```
10 INPUT a
20 INPUT b
30 PRINT (a AND a>=b)+(b AND a<b)
40 GO TO 10
```

Ogni volta stampa il più grande dei due numeri **a** e **b**; convincetevi che potete pensare

x AND y

come se fosse

x se **y** (altrimenti il risultato è 0)

e

x OR y

come se fosse

x se non **y** (nel qual caso il risultato è 1).

Una qualunque espressione che adoperi **AND** o **OR** in questo modo, è chiamata un'espressione logica. Per esempio **OR** potrebbe essere usato così:

```
LET prezzo totale=prezzo senza tasse*(1.15 OR v$="iva esente")
```

Notate come **AND** tenda a combinarsi con le somme, dato che può essere 0 e **OR** tenda a combinarsi con le moltiplicazioni, dato che può essere 1.

Anche con le stringhe si possono costruire espressioni logiche, usando solo **AND**:

x$ AND y ha il valore: $\begin{cases} \text{x\$ se } y \text{ è "non 0"} \\ \text{" " se } y \text{ è "0"} \end{cases}$

che significa **x$** se **y** (altrimenti stringa nulla).

Provate il seguente programma, che legge due stringhe e le mette in ordine alfabetico:

```
10 INPUT "scrivete due stringhe",a$,b$
20 IF a$>b$ THEN LET c$=a$: LET a$=b$: LET b$=c$
30 PRINT a$;" ";("<" AND a$<b$)+("=" AND a$=b$);" ";b$
40 GO TO 10
```

Esercizi

1. A volte il BASIC si comporta in modo differente dalle regole di grammatica, per quanto vi sia sempre molto simile. Considerate per esempio la frase "Se *a* non è uguale a *b* o a *c*" e cercate di scriverla in BASIC. Sappiate che la risposta non è
IF A<>B OR C
e neppure
IF A<>B OR A<>C

CAPITOLO 23

Il Set di Caratteri

Sommario

CODE, CHR$
POKE, PEEK
USR
BIN

Le lettere, le cifre, i segni di punteggiatura, ecc., che appaiono nelle stringhe, sono chiamati *caratteri*, e formano l'alfabeto dello ZX Spectrum, chiamato *set di caratteri*. La maggior parte di questi caratteri sono simboli, ma ne esistono alcuni, chiamati *tokens*, che rappresentano intere parole, come **PRINT, STOP,** <> e così via.

Esistono 256 caratteri, ognuno dei quali ha un codice da 0 a 255: ne troverete una lista completa nell'Appendice A. Per convertire da codice a carattere vi sono due funzioni: **CODE** e **CHR$**.

CODE si applica ad una stringa, e ritorna il primo carattere di essa, oppure 0 se la stringa è vuota.

CHR$ si applica un numero, e ritorna il carattere con il codice corrispondente al numero dato. Il seguente programma stampa tutto il set di caratteri:

```
10 FOR a=32 TO 255: PRINT CHR$a;:NEXT a
```

All'inizio potrete osservare uno spazio, 15 simboli e segni di punteggiatura, successivamente le 10 cifre, 7 altri simboli, le lettere maiuscole, 6 altri simboli, le lettere minuscole ed infine 5 altri simboli. Tutti questi (tranne £ e ©) sono tratti da un set di caratteri standard molto diffuso, conosciuto come ASCII (per American Standard Codes for Information Interchange); lo ZX Spectrum usa anche i codici che lo standard ASCII attribuisce a questi caratteri.

La rimanente parte di caratteri non è appartenente allo standard ASCII, ma è proprio dello ZX Spectrum. I primi di questi sono lo spazio e i 15 simboli formati da quadratini bianchi e neri che vengono chiamati simboli *grafici*, e possono essere usati per disegnare delle figure. I simboli grafici possono essere introdotti direttamente dalla tastiera, usando il modo grafico. Per entrare nel modo grafico, basta premere **GRAPHICS (CAPS SHIFT** e **9**), ed il cursore cambia in **G** . Quando siete nel modo grafico, le cifre dall'1 all'8 ritornano i simboli grafici, rappresentati sui rispettivi tasti, e, con lo **SHIFT** premuto contemporaneamente, ritornano lo stesso simbolo, invertito, cioè col bianco che diventa nero e viceversa.

Sia che abbiate premuto lo **SHIFT** o no, la cifra **9** vi riporta al modo normale (L), e la cifra 0 opera come **DELETE**.

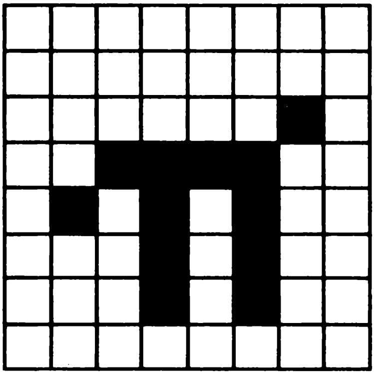
Ecco la lista dei 16 caratteri grafici:

Simbolo	Codice	Si ottiene con	Simbolo	Codice	Si ottiene con
	128	G 8	█	143	G shifted 8
▝	129	G 1	▙	142	G shifted 1
▘	130	G 2	▟	141	G shifted 2
▀	131	G 3	▄	140	G shifted 3
▗	132	G 4	▛	139	G shifted 4
▐	133	G 5	▌	138	G shifted 5
▚	134	G 6	▞	137	G shifted 6
▜	135	G 7	▖	136	G shifted 7

Dopo i simboli grafici, vedrete apparire un'altra copia dell'alfabeto, dalla A alla U. Questi caratteri possono essere ridefiniti da voi, anche se all'accensione, sono inizializzati come lettere. Si chiamano *caratteri definiti dall'utente*. Potete introdurre questi caratteri nel calcolatore direttamente dalla tastiera, usando le lettere da A a U nel modo grafico.

Per capire come definire un carattere a vostro piacimento, seguite l'esempio: definisce un carattere per stampare π.

(i) Disegnate il carattere così come deve essere stampato. Tenete presente che ogni carattere ha a sua disposizione una matrice di 8*8 punti, ognuno dei quali può essere o nel colore della carta, o nel colore dell'inchiostro (consultate i primi capitoli). Fate un disegno analogo al seguente, mettendo nei quadratini neri quello che volete, con i colori dell'inchiostro:



Fate bene attenzione che è stato lasciato un margine di un quadratino bianco intorno al bordo, dato che tutte le lettere ne hanno uno (ad eccezione delle lettere minuscole col gambo, che arrivano giù fino in basso).

(ii) Decidete quale carattere volete che riporti il vostro carattere grafico; per esempio, in questo caso, useremo la P, in modo tale che se premiamo P in modo grafico, otteniamo π.

(iii) Memorizzate la nuova matrice di punti. Ogni carattere definito dall'utente ha la sua matrice di punti memorizzata come 8 numeri, uno per ogni riga. Potete scrivere ognuno di questi numeri con il comando **BIN** seguito da 8 zeri o uno (0 per la carta, 1 per l'inchiostro) in questo modo gli otto numeri del carattere π sono:

BIN 00000000
BIN 00000000
BIN 00000010
BIN 00111100
BIN 01010100
BIN 00010100
BIN 00010100
BIN 00010100
BIN 00000000

Se conoscete i numeri binari, vi sarà utile sapere che **BIN** è usato per scrivere un numero in forma binaria invece che in forma decimale.

Questi 8 numeri sono conservati in memoria in 8 differenti posizioni, ognuna delle quali ha ovviamente un indirizzo. L'indirizzo del primo byte, o primo gruppo di otto bit, è **USR "P"** (P perchè è la lettera che abbiamo scelto al punto (ii)). Quello del secondo sarà **USR "P"+1**, e così via fino all'ottavo, che avrà indirizzo **USR "P"**+7.

USR è una funzione che converte un argomento stringa nell'indirizzo del primo

byte della memoria per il corrispondente carattere definito dall'utente. L'argomento stringa deve essere un singolo carattere, che può essere sia il carattere grafico definito precedentemente dall'utente, o la corrispondente lettera maiuscola o minuscola, appunto. **USR** può anche essere usato con argomento numerico, e allora assume un altro significato, che tratteremo più avanti.

Se non siete sicuri di aver capito, non preoccupatevi: il seguente programma vi chiarirà le idee, memorizzando il nostro carattere:

```
10 FOR n=0 TO 7
20 INPUT riga: POKE USR "P"+n,riga
30 NEXT n
```

Si fermerà otto volte per permettervi di introdurre gli otto **BIN ...** scritti sopra; scriveteli nel giusto ordine, iniziando col primo.

Il comando **POKE** usato nel programma memorizza un numero direttamente in una posizione di memoria, prevaricando il meccanismo usuale del BASIC. Il contrario di **POKE** è **PEEK**, che vi permette di conoscere il contenuto di una posizione di memoria, senza modificarlo. Queste funzioni saranno approfondite nel capitolo 33.

Ritornando al set dei caratteri, dopo i caratteri definiti dall'utente, vengono le parole chiave.

Notate che non sono stati stampati i primi 32 caratteri (che hanno codice da 0 a 31); questi sono caratteri di controllo e non producono nessun simbolo stampabile, ma hanno qualche effetto magari non immediatamente visibile sul video, oppure sono usati per controllare qualcosa che non sia lo schermo, e quindi quest'ultimo stampa un **?** perchè non li capisce. Troverete una descrizione accurata di questi caratteri nell'appendice A.

Si possono inviare al video tre caratteri: i loro codici sono 6, 8 e 13; ma alla fine, **CHR$ 8** sarà il solo carattere che troverete utile.

CHR$ 6 stampa gli spazi esattamente allo stesso modo della virgola in un comando di **PRINT**, quindi:

```
PRINT 1; CHR$ 6; 2
```

funziona allo stesso modo che

```
PRINT 1,2
```

Ovviamente questo non è un modo molto intelligente di usarlo; più appropriatamente:

```
LET a$="1"+CHR$ 6+"2"
PRINT a$
```

CHR$8 è il "backspace", e muove il cursore indietro di una posizione provate:

```
PRINT "1234"; CHR$ 8;"5"
```

che stampa alla fine **1235**.

CHR$ 13 è “newline”, e sposta il cursore all’inizio della linea seguente.

Il video usa anche i caratteri con codice da 16 a 23 questi sono spiegati nei Capitoli 24 e 25. Tutti i caratteri di controllo sono comunque listati nell’Appendice A.

Usando i codici per i caratteri possiamo estendere il concetto di ordine alfabetico, per comprendere le stringhe composte da caratteri anche non letterali. Se invece di pensare in termini di un alfabeto di 26 lettere, utilizziamo l’alfabeto dello ZX Spectrum di 256 caratteri, ordinato per codice, possiamo ordinare alfabeticamente qualunque stringa. Per esempio, le seguenti stringhe sono nell’ordine alfabetico dello ZX Spectrum. (Notate che, piuttosto stranamente, le lettere minuscole vengono dopo le maiuscole, quindi “a” viene dopo “Z”, e contano anche gli spazi).

CHR$ 3+“GIARDINI ZOOLOGICI”
CHR$ 8+“ABACO”
“ AAAARGH!”
“(Parentesi)”
“100”
“1300 ivato”
“ABACO”
“Abaco”
“PRINT”
“Zoo”
“[interpolazione]”
“abaco”
“amaca”
“zoo”
“zoologia”

Ecco la regola che determina quale, fra due stringhe, venga prima in ordine alfabetico. Primo, si confronta il primo carattere: se sono diversi, allora la prima stringa sarà quella dove il codice del primo carattere sarà più basso dell’altra. Se i primi caratteri delle due stringhe sono uguali, si ripete l’operazione per i secondi, e così via, fino al termine: se una stringa finisce prima dell’altra, allora sarà la prima.

Le relazioni =, <, <=, >= e <> sono valide per le stringhe come per i numeri, ed applicano automaticamente la regola; < significa “viene prima di”, >“viene dopo di”, così

“AA uomo”<“ABACO”
“ABACO”>“AA uomo”

sono entrambe vere.

<= e >= operano allo stesso modo che con i numeri, così

"La stessa stringa"<="La stessa stringa"

è vero,ma

"La stessa stringa"<"La stessa stringa"

è falso.

Esercitatevi su questi argomenti per mezzo del programma riportato di seguito, che legge e ordina due stringhe.

```
10 INPUT "Introducete due stringhe:",a$,b$
20 IF a$>b$ THEN LET c$=a$: LET a$=b$: LET b$=c$
30 PRINT a$;" ";
40 IF a$<b$ THEN PRINT "<": GO TO 60
50 PRINT "="
60 PRINT " ";b$
70 GO TO 10
```

Notate come alla linea 20 si usi **c$** per scambiare **a$** con **b$**

LET a$=b$: LET b$=a$

non produrrebbe l'effetto desiderato: cercate di capire il perchè.

Il seguente programma definisce dei caratteri definiti dall'utente, per mostrarvi i pezzi degli scacchi:

P per pedone
T per torre
C per cavallo
A per alfiere
K per re
Q per regina

Pezzi degli scacchi

```
 5 LET b=BIN 01111100: LET c=BIN
   00111000: LET d=BIN 00010000
10 FOR n=1 TO 6: READ p$: REM 6 pezzi
20 FOR f=0 TO 7: REM legge i pezzi in 8 bytes
30 READ a: POKE USR p$+f,a
40 NEXT f
50 NEXT n
```

```
100 REM alfiere
110 DATA "a",0,d, BIN 00101000, BIN 01000100
120 DATA BIN 01101100,c,b,0
130 REM re
140 DATA "k",0,d,c,d
150 DATA c, BIN 01000100,c,0
160 REM torre
170 DATA "t",0, BIN 01010100,b,c
180 DATA c,b,b,0
190 REM regina
200 DATA "q",0,BIN 01010100, BIN 00101000,d
210 DATA BIN 01101100,b,b,0
220 REM pedone
230 DATA "p",0,0,d,c
240 DATA c,d,b,0
250 REM cavallo
260 DATA "c",0,d,c, BIN 01111000
270 DATA BIN 00011000,c,b,0
```

Notate che 0 può essere usato al posto di **BIN 00000000**.
Dopo lo svolgimento del programma, osservate i pezzi nel modo grafico.

Esercizi

1. Immaginatevi lo spazio riservato ad un simbolo diviso in quattro parti come una finestra se ogni quarto può essere o bianco o nero, allora vi saranno 2*2*2*2=16 possibili combinazioni di colore.
Individuatele tutte nel set dei caratteri.

2. Fate girare questo programma:

```
10 INPUT a
20 PRINT CHR$ a;
30 GO TO 10
```

Se lo provate, troverete che **CHR$ a** è *arrotondato* al più vicino numero intero e se il valore di a non è compreso tra 0 e 255, il programma si interromperà dando il messaggio **B integer out of range**.

3. Qual'è la minore fra le due

"DEMONE"
"demone"

4. Studiate quale modifiche sarebbero necessarie al programma per la grafica definita dall'utente, in modo da poter usare **READ** e **DATA** al posto delle linee di **INPUT**.

CAPITOLO 24

Uso avanzato di PRINT e INPUT

Sommario
CLS
PRINT elementi: nessun elemento
Espressioni (di tipo numerico o stringa):
TAB espressione numerica,
AT espressione numerica, espressione numerica
PRINT separatori: , ; '
INPUT elementi: variabili (tipo numerico o stringa)
LINE variabile stringa
Elementi di **PRINT** che non inizino con una lettera.
(Le parole chiave vengono considerate come non inizianti con una lettera).
Scrolling

Se avete seguito i precedenti capitoli del libro, avrete usato **PRINT** molte volte, e vi sarete quindi fatti un'idea di come funziona. Le espressioni usate per stampare dei valori sono chiamate *elementi* di una **PRINT**, e sono separate da virgole o punti e virgole, che si chiamano *separatori*. Un elemento della **PRINT** può anche non essere niente, il che significa che si possono usare due virgole, una in fila all'altra.

Ci sono altri due elementi della **PRINT** che non indicano al calcolatore cosa stampare, ma dove. Per esempio, **PRINT AT 11,16;** "*" stampa un asterisco nel centro dello schermo.

AT linea, colonna

sposta la posizione di stampa (il posto dove sarà stampato il prossimo elemento), alla linea ed alla colonna specificate. Le linee sono numerate da 0 a 21 (dall'alto in basso), e le colonne da 0 a 31 (da sinistra a destra).

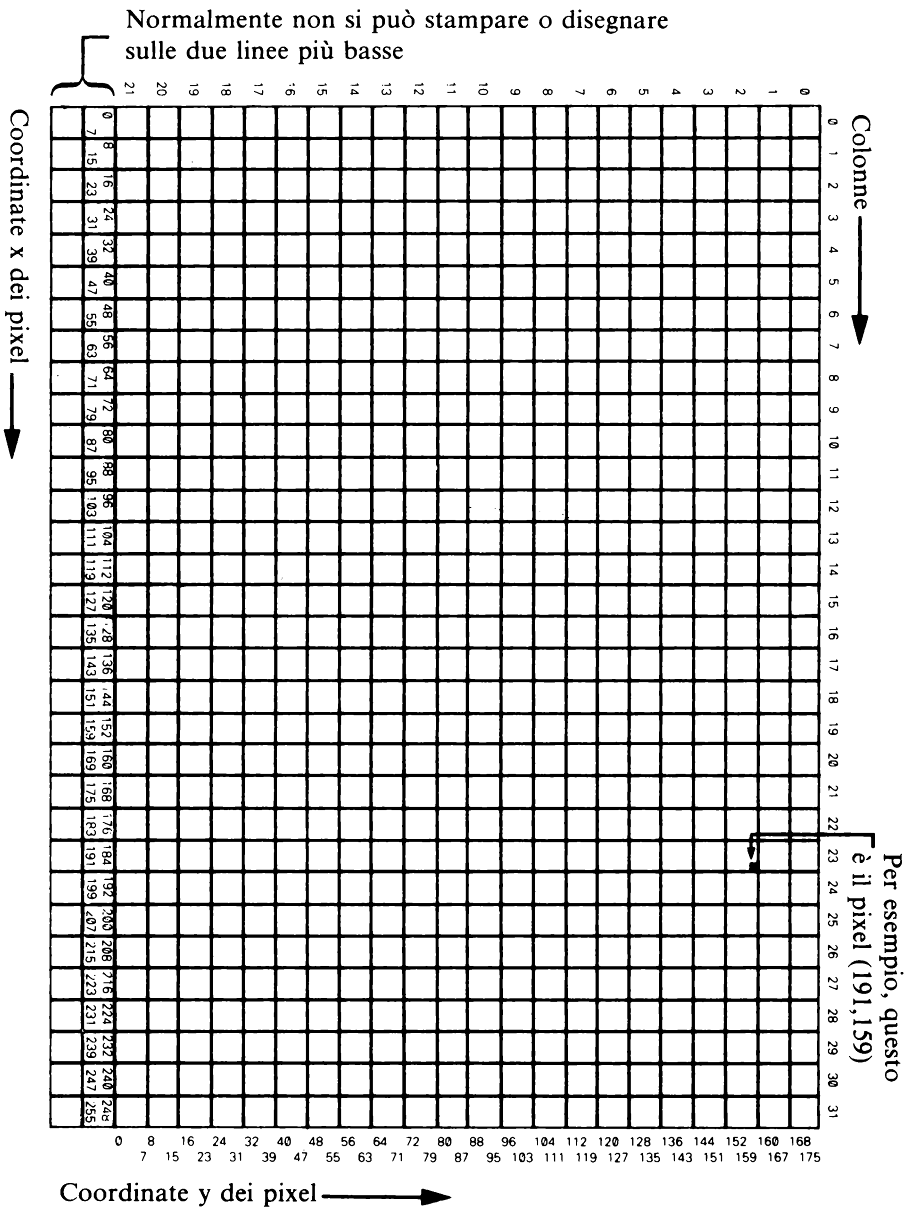
TAB colonna

stampa gli spazi che servono per spostare la posizione di stampa alla colonna specificata. **TAB** rimane sulla stessa linea dove si trova il cursore, a meno che la posizione di stampa specificata non sia precedente alla posizione di stampa attuale, nel qual caso si sposta alla linea successiva. Notate che il calcolatore riduce il numero di colonne in "modulo 32" (le divide per 32 e prende il resto); cosicchè **TAB 33** è equivalente a **TAB 1**.

Osservate l'esempio:

PRINT TAB 30; 1; TAB 12; "Indice"; AT 3,1; "CAPITOLO"; TAB 24; "pagina"

illustra come potreste stampare l'intestazione dell'indice di un libro.


Normalmente non si può stampare o disegnare sulle due linee più basse
Colonne
Coordinate x dei pixel
Per esempio, questo è il pixel (191,159)
Coordinate y dei pixel
0 8 16 24 32 40 48 56 64 72 80 88 96 104 112 120 128 136 144 152 160 168
7 15 23 31 39 47 55 63 71 79 87 95 103 111 119 127 135 143 151 159 167 175

Provate a far girare

10 FOR n=0 TO 20
20 PRINT TAB 8*n;n;
30 NEXT n

Questo programma vi illustra chiaramente cosa significa la riduzione in modulo 32 dei numeri di **TAB**.

Per avere un esempio più elegante, cambiate l'8 della linea 20 in 6. Tenete presente anche le seguenti osservazioni:

(i) Gli elementi di stampa dopo **TAB** o **AT** sono usualmente terminati con un punto e virgola, come nel programma esempio. Questo perchè se usate una virgola oppure niente, si ottiene il non sempre desiderabile risultato che la posizione di stampa, dopo essere stata accuratamente posizionata, venga spostata.

(ii) Non è possibile stampare sulle due linee più basse dello schermo, la 22 e la 23, dato che sono riservate per i comandi, per la lettura dei dati, per i messaggi e così via. Quando si parla della linea bassa dello schermo, solitamente si intende la linea 21.

(iii) È ovviamente possibile usare **AT** per posizionare la posizione di stampa anche dove c'è già qualcosa di stampato sullo schermo; le precedenti stampe saranno cancellate da quelle nuove.

CLS pulisce tutto lo schermo, funzione svolta anche dai comandi **CLEAR** e **RUN**, che eseguono però anche altre funzioni.

Quando il calcolatore, stampando, raggiunge l'ultima riga bassa dello schermo, esegue lo *scrolling*, ovvero sposta tutte le linee stampate in su di una per fare posto a quella nuova, analogamente alla macchina da scrivere, ma cancellando la prima linea in alto. Per osservare ciò eseguite:

CLS: FOR n=1 TO 22: PRINT n: next n

e poi

PRINT 99

diverse volte.

Se il calcolatore sta stampando qualcosa in un ciclo, non cancella niente dal video senza che voi abbiate avuto il tempo di esaminarlo attentamente. Provate

CLS: FOR n=1 TO 100: PRINT n: NEXT n

Quando il calcolatore ha riempito completamente lo schermo, si ferma scrivendoci in basso **scroll?**, dandovi quindi il tempo di esaminare con calma le prime 22 linee di stampato. Quando avrete terminato, premete **y** (per "yes", sì), ed il calcolatore riempirà un altro schermo di stampato. Invece di **y** potete usare qualunque tasto, eccetto **n** (per "no"), **STOP (SYMBOL SHIFT** ed **a**), o **SPACE** (il tasto **BREAK**), che causano l'arresto del programma e la stampa del messaggio **D BREAK - CONT repeats.**

Il comando **INPUT** può fare molto più di quanto non abbiate visto finora. Avrete già incontrato messaggi di **INPUT** tipo

INPUT "Quanti anni hai?"; anni

nei quali il calcolatore stampa il messaggio **Quanti anni hai?**, in basso allo schermo, ed aspetta che voi scriviate la vostra età.

Una linea di **INPUT** è composta da una serie di elementi e di separatori, che hanno la stessa funzione che in una linea di **PRINT; Quanti anni hai?** e **anni** sono entrambi elementi della **INPUT**. Gli elementi che potete introdurre in una **INPUT** sono gli stessi usati dalle **PRINT**, ma vi sono alcune importanti differenze.

La differenza più importante della **INPUT** è la variabile da leggere da tastiera, **anni** nell'esempio. In effetti **INPUT** considera ogni elemento che inizia con una lettera come variabile da assegnare.

Differentemente da come si potrebbe aver capito, si possono stampare anche valori di variabili come parti di un messaggio; è sufficiente racchiudere il nome della variabile da stampare tra parentesi. Qualunque espressione che inizia con una lettera, e che deve essere stampata come parte di un messaggio, deve essere racchiusa tra parentesi.

Tutti gli altri elementi della **PRINT** che non sono soggetti a queste regole, possono anche essere elementi della **INPUT**. Il seguente esempio illustra le regole appena viste:

LET i miei anni = INT (RND*100): INPUT("io ho"; i miei anni; "anni."); "Quanti anni hai? ",i tuoi anni

i miei anni è tra parentesi, quindi il suo valore viene stampato, mentre **i tuoi anni** non è fra parentesi e quindi il suo valore viene letto da tastiera.

Qualunque cosa scritta da un comando di **INPUT** compare nella parte bassa dello schermo destinata all'input, che si comporta, in qualche modo, indipendentemente dalla parte alta destinata all'output. Si noti che le sue linee sono numerate alla prima linea della sezione bassa, anche se questa si è spostata rispetto alla mezzeria dello schermo, a causa dello scrolling.

Provate il seguente esempio per vedere come funziona **AT**, in un comando di **INPUT**:

10 INPUT "Questa è la linea 1.",a$; AT 0,0; "Questa è la linea 0.",a$; AT 2,0; "Questa è la linea 2.",a$; AT 1,0; "Questa è ancora la linea 1.",a$

(premete **ENTER** ogni volta che il programma si ferma). Quando viene stampato il messaggio **Questa è la linea 2**, la parte bassa dello schermo si muove per fargli posto,facendo si che anche la numerazione delle linee si muova, in modo tale che le linee di testo mantengano i numeri originali. Provate:

```
10 FOR n=0 TO 19: PRINT AT n,0; n; : NEXT n
20 INPUT AT 0,0; a$; AT 1,0; a$; AT 2,0; a$; AT 3,0; a$; AT 4,0; a$;
   AT 5,0; a$;
```

Durante la fase di salita della parte bassa dello schermo, la parte alta rimane immobile fino a quando la cima della parte bassa non riaggiunge la posizione di stampa: allora, tutto lo schermo esegue lo scrolling. Si può evidenziare questo fenomeno usando due colori diversi per **PAPER** e **BORDER**: provate.

Esiste anche un'altra modalità per la lettura delle variabli stringa, che consiste nel scrivere la parola chiave **LINE** dopo la **INPUT** e prima della variabile stringa da leggere, come in

INPUT LINE a$

In questo caso il calcolatore non stamperà gli apici di stringa che normalmente stampa quando legge una variabile stringa, anche se si comporta come se ci fossero, così, se scrivete

gatto

come variabile di **INPUT**, **a$** assumerà il valore **gatto**. Dato che mancano gli apici di stringa sullo schermo, non è possibile cancellarli per scrivere qualche altro tipo di espressione stringa per la **INPUT**. Ricordatevi che **LINE** non può essere usato per le variabili numeriche.

I caratteri di controllo **CHR$ 22** e **CHR$ 23** funzionano in modo abbastanza simile a **AT** e **TAB**. Funzionano in modo apparentemente strano, dato che entrambi devono essere seguiti da due o più caratteri, che non hanno il solito significato: sono infatti trattati come numeri (cioè col valore del loro codice), per specificare la linea e la colonna (per **AT**), o la posizione di **TAB** (per **TAB**). Nella maggior parte dei casi, è più comodo usare **AT** e **TAB**, come già visto, ma, in qualche caso particolare, i caratteri di controllo possono rivelarsi particolarmente utili. Il carattere di controllo per **AT** è **CHR$ 22**. Il primo carattere che lo segue specifica il numero di linea, e il secondo il numero di colonna, cosicchè

PRINT CHR$ 22+CHR$ 1+ CHR$ c;

è esattamente analogo a

PRINT AT 1,c;

Notate che **CHR$ 1** e **CHR$ c (c=13)**, normalmente avrebbero un significato ben diverso. Dopo **CHR$ 22** perdono il loro significato originale.

Il carattere di controllo equivalente a **TAB** è **CHR$ 23**, e i due caratteri che lo seguono sono usati per indicare un numero compreso tra 0 e 65535, che specifica il numero di **TAB** specificati come dopo un **TAB**.

```
PRINT CHR$ 23+CHR$ a+CHR$ b;
```

provoca esattamente lo stesso effetto di

```
PRINT TAB a+256*b
```

Se non volete che la macchina scriva sempre **scroll?** alla fine del video, potete usare

```
POKE 23692,255
```

di tanto in tanto. Dopo questa linea, il calcolatore riempirà 255 schermi prima di chiedere lo **scroll?**. Per esempio, provate:

```
10 FOR n=0 to 10000
20 PRINT n: POKE 23692,255
30 NEXT n
```

e vedrete le scritte volare sullo schermo!

Esercizi

Fate girare questo programma per verificare la vostra conoscenza della tabellina della moltiplicazione:

```
 10 LET m$=""
 20 LET a=INT (RND*12)+1: LET b=INT (RND*12)+1
 30 INPUT (m$) ' ' "cosa fa"; (a); "*"; (b); "?"; c
100 IF c=a*b THEN LET m$="Esatto.": GO TO 20
110 LET m$="Errato. Riprova.": GO TO 30
```

Se avete qualche difficoltà, vi farà piacere sapere che non dovete fare i conti: infatti, se il calcolatore vi chiede cosa dà 2*3, basta che scriviate **2*3**.

Per evitare questo trucco, potete leggere delle stringhe invece che dei numeri quindi sostituite **c$** al posto di **c** alla linea 30, e **VAL c$** al posto di **c** alla linea 100, e inserite la linea 40:

```
40 IF c$<>STR$ VAL c$ THEN LET m$="Rispondi propriamente, con
   un numero.": GO TO 30
```

Anche così, però, non siete completamente sicuri. Potreste accorgervi, dopo qualche ragionamento, che se cancellate le virgolette e scrivete **STR$ (2*3)**, non dovrete più fare i calcoli. Per mettervi completamente a riparo da ogni possibilità d'imbroglio sostituite alla linea 30 **c$** con **LINE c$**. Provate a far girare questo programma davanti ad un bambino che deve imparare le tabelline: se gli spiegate qualcosa del calcolatore, molto probabilmente scoprirà anche lui il trucco.

CAPITOLO 25

Colori

Sommario
INK, PAPER, FLASH, BRIGHT, INVERSE, OVER
BORDER

Scrivete e fate girare il seguente programma:

```
 10 FOR m=0 TO 1: BRIGHT m
 20 FOR n=1 TO 10
 30 FOR c=0 TO 7
 40 PAPER c: PRINT "    ";: REM 4 spazi colorati
 50 NEXT c: NEXT n: NEXT m
 60 FOR m=0 TO 1 BRIGHT m: PAPER 7
 70 FOR c=0 TO 3
 80 INK c: PRINT c;"   ";
 90 NEXT c: PAPER 0
100 FOR c=4 TO 7
110 INK c: PRINT c;"   ";
120 NEXT c: NEXT m
130 PAPER 7: INK 0: BRIGHT 0
```

Questo vi mostra gli otto colori (che includono anche il bianco e il nero), e i due livelli di luminosità che lo ZX Spectrum è in grado di produrre su un video a colori. Se la vostra televisione è in bianco e nero, vedrete soltanto diverse tonalità di grigio. Ecco una lista dei colori, nell'ordine in cui sono riportati sui tasti numerici:

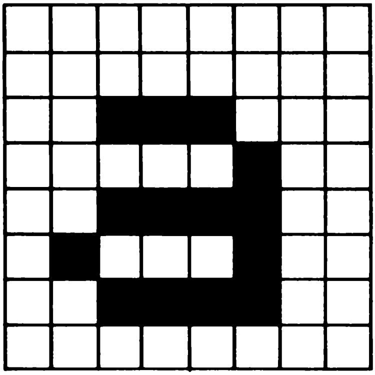
0 - nero
1 - blu
2 - rosso
3 - porpora, o magenta
4 - verde
5 - blu chiaro, o ciano
6 - giallo
7 - bianco

In una televisione in bianco e nero, i numeri corrispondono a una diversa gradazione di grigio, in ordine dal più scuro al più chiaro.

Per una completa familiarità con l'uso dei colori, è necessaria la vostra conoscenza dell'organizzazione dello schermo.

Esso è diviso in 768 (24 linee di 32 caratteri) posizioni dove i caratteri possono essere stampati: ognuno di essi è rappresentato da un quadrato di punti con dimensione 8*8, come la a in figura.

Vi ricorderà sicuramente i caratteri definiti dall'utente, già visti nel capitolo 23, dove usavate uno 0 per ogni quadratino bianco e un 1 per ogni nero.



Ogni carattere è anche associato a due colori: il colore dell'inchiostro, o il colore di stampa, cioè quello dei quadratini neri della nostra **a**, ed il colore della carta, o colore dello sfondo, che è quello dei quadratini bianchi. Il colore iniziale dell'inchiostro è il nero, quello della carta è il bianco, cosicchè appare scritto proprio come nella figura.

Ogni posizione ha anche qualcosa che ne indica la luminosità, normale o extra luminoso, e qualcosa che deve indicare se deve lampeggiare o no. Il lampeggiamento si può ottenere invertendo continuamente il colore dell'inchiostro e della carta. Tutto ciò è codificato in numeri, ne consegue che ogni posizione ha
(i) un quadrato di 8*8, formato da zeri e uno, per definire la sagoma del carattere, dove 0 sta per la carta e 1 per l'inchiostro.
(ii) i colori della carta e dell'inchiostro, codificati ognuno con un numero da 0 a 7.
(iii) una diversa luminosità, 0 per normale e 1 per extra.
(iv) la possibilità di lampeggiamento, 0 no e 1 si.

Il colore, la luminosità e l'indicazione di lampeggiamento per una certa posizione sono chiamati *attributi*. Notate che, dato che gli attributi coprono un'intera posizione di carattere, non è possibile avere più di due colori in una posizione dato di 64 punti.

Quando stampate qualcosa sullo schermo, modificate lo schema dei punti e gli attributi di quella posizione. All'inizio non notate alcuna differenza, dato che tutto viene stampato con inchiostro nero su carta bianca, a luminosità normale e senza lampeggiamento. Potete variare gli attributi a vostro piacimento con i comandi **INK, PAPER, BRIGHT** e **FLASH**. Provate

PAPER 5

e poi scrivete qualcosa la vedrete apparire su uno sfondo ciano, dove man mano che i caratteri sono stampati lo sfondo della loro posizione assume il colore ciano (di codice 5).

Gli altri comandi lavorano in modo analogo, così

PAPER numeri compresi tra 0 e 7
INK numeri compresi tra 0 e 7

BRIGHT 0 oppure 1 (0 non attivo, 1 attivo)
FLASH 0 oppure 1 (0 non attivo, 1 attivo)

ogni posizione occupata da un nuovo carattere assumerà gli attributi che avete settato. Provate a vostro piacimento, e tenendo presente che uno spazio è un carattere che ha lo stesso colore per **INK** e **PAPER**, cercate di esaminare il funzionamento del programma esempio.

I parametri possono avere anche qualche valore non specificato in tabella.

8 può essere usato su tutti i 4 comandi, e significa trasparente, dato che non altera l'attributo della posizione quando viene stampato un carattere. Supponete per esempio di scrivere

PAPER 8

ovviamente nessun carattere avrà colore 8, dato che non ne esistono di quel codice: succederà invece che quando un carattere è stampato, lo sfondo rimane dello stesso precedente colore. **INK 8, BRIGHT 8** e**FLASH 8** lavorano allo stesso modo.

9 può essere usato solo con **PAPER** e **INK**, e significa contrasto. Il colore dell'inchiostro o della carta, a seconda del comando che avete usato, viene fatto ricontrastare con l'altro, venendo posto bianco su un colore scuro (nero, blu, rosso, magenta), e nero contro un colore chiaro (verde, ciano, giallo o bianco).

Provatelo scrivendo

INK 9: FOR c=0 TO 7: PAPER c: PRINT c: NEXT c

Potete provare un esempio più appariscente della potenza di questo comando facendo eseguire il comando riportato sotto dopo aver disegnato delle strisce colorate sullo schermo con il programma riportato all'inizio del capitolo.

**INK 9: PAPER 8: PRINT AT 0,0;: FOR n=1 TO 1000: PRINT n;:
NEXT n**

In questo esempio il colore dell'inchiostro è sempre fatto contrastare col vecchio colore che la carta aveva in ogni posizione.

Provate anche a mettere in modo contrasto sia **PAPER** che **BORDER** ed a stampare qualcosa con diversi valori di **PAPER** e **INK** come elementi della **PRINT**. Sapete pensare qualche applicazione?

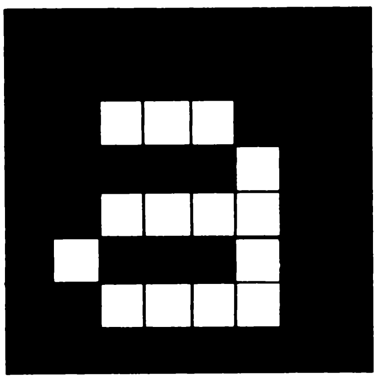
La televisione a colori è costruita sulla particolarità dell'occhio umano di vedere ogni colore come composto da tre colori, ovvero i tre colori primari blu, rosso e verde. Gli altri colori sono ottenuti come un misto di questi; per esempio, il magenta è ottenuto mescolando il rosso col blu, ecco quindi perchè il suo codice è 3, cioè la somma dei codici del blu e del rosso.

Per immaginarvi da dove provengono tutti gli otto colori, pensate a tre macchie di luce rettangolari colorate una blu, l'altra rossa, l'altra ancora verde, proiettate da vicino su un pezzo di carta bianca posto al buio. Dove le tre luci si sovrappongono, vedrete un gran misto di colori, come mostrato in questo programma (notate che gli spazi incolore sono anche ottenuti con **SHIFT** e **8** nel modo grafico):

```
  10 BORDER 0: PAPER 0: INK 7: CLS
  15 LET a$ = "■■■■■■■■■■": REM 10 quadratini d'inchiostro
  20 FOR a=1 TO 6
  30 PRINT TAB ø; INK 1; a$; INK 2; a$; INK 4; a$
  40 NEXT a
  50 LET dataline=200
  60 GO SUB 1000
  70 LET dataline=210
  80 GO SUB 1000
  90 STOP
 200 DATA 5, 3, 7, 5, 6, 3
 210 DATA 6, 5, 6, 4, 7, 5
1000 FOR a=1 TO 6
1010 RESTORE dataline
1020 FOR b=1 TO 6
1030 READ c: PRINT INK c; "■■■■■";: REM 5 quadratini d'inchiostro
1040 NEXT b: PRINT : NEXT a
1050 RETURN
```

Esiste anche una funzione chiamata **ATTR**, che ritorna gli attributi di una data posizione dello schermo. Dato che è una funzione molto complessa, sarà descritta al termine del capitolo.

Esistono altri due comandi, **INVERSE** e **OVER**, che controllano non gli attributi, ma la matrice di punti stampata sullo schermo. Usano come parametri solo i numeri 0 e 1 esattamente come **FLASH** e **BRIGHT**. Se eseguite **INVERSE 1**, la matrice di punti che verrà stampata sarà esattamente l'inverso della matrice normale: il colore della carta sarà usato per il colore dell'inchiostro e viceversa. Quindi la **a** dell'esempio precedente verrebbe stampata in questo modo:



Se avevamo gli attributi settati dall'accensione, cioè inchiostro nero e carta bianca, allora la a apparirà bianco su nero. La matrice di punti è stata cambiata, ma gli attributi sono ancora gli stessi.

Il comando **OVER 1** attiva un particolare tipo di sovraimpressione. Di solito, quando qualcosa viene stampato in una posizione, il carattere che vi era precedentemente viene totalmente cancellato. Adesso, il nuovo carattere verrà sommato, o meglio, sovrapposto al vecchio (osservate l'esercizio 3). Questa caratteristica può essere molto utile per scrivere caratteri composti, tipo lettere con accento, o lettere tedesche, come nel programma che segue, che stampa una "o" con la umlaut:

```
10 OVER 1
20 FOR n=1 TO 32
30 PRINT "o"; CHR$ 8;"""";
40 NEXT n
```

(notate il carattere di controllo, **CHR$** 8, che sposta la posizione di stampa indietro di un carattere).

INK, PAPER, ecc., possono essere usati in modo più pratico, inseriti cioè come elementi in un comando **PRINT**, seguiti dal punto e virgola. Avranno esattamente lo stesso effetto che avrebbero avuto se fossero stati scritti da soli, in un comando a parte, la differenza sta nel fatto che ora avranno un effetto temporaneo: varranno cioè solo per quegli elementi stampati dalla **PRINT**. Quindi, se scrivete

PRINT PAPER 6; "x";: PRINT"y"

solo la x sarà su sfondo giallo.

INK e gli altri comandi, usati come comandi, non modificano i colori della parte bassa dello schermo; quest'ultima usa il colore del bordo come colore della carta, e il codice 9 per contrastare il colore dell'inchiostro, non ha niente che lampeggia, e tutto è a luminosità normale. Il bordo può assumere uno qualunque degli otto colori normali (non 8 e 9), tramite il comando

BORDER colore

Quando introducete i dati richiesti da una **INPUT**, appaiono, secondo la regola, contrastati rispetto al colore del bordo. È invece possibile cambiare i colori del messaggio scritto dal calcolatore usando **INK, PAPER**, ecc., come elementi nel comando di **INPUT**, esattamente come avreste fatto in un comando di **PRINT**. Anche in questo caso il loro effetto è momentaneo, e dura fino alla fine del comando, o fino a che non viene introdotto qualche dato, secondo quello che viene prima. Provate

INPUT FLASH 1; INK 1; "Scrivete il vostro numero di telefono"; n

Esiste un altro modo di cambiare i colori usando i caratteri di controllo, analogamente a quanto visto per **AT** e **TAB** nel capitolo 24.

CHR$ 16 corrisponde a **INK**
CHR$ 17 corrisponde a **PAPER**
CHR$ 18 corrisponde a **FLASH**
CHR$ 19 corrisponde a **BRIGHT**
CHR$ 20 corrisponde a **INVERSE**
CHR$ 21 corrisponde a **OVER**

Tutti questi caratteri di controllo sono seguiti da un carattere che indica un colore tramite il suo codice, quindi, per esempio:

PRINT CHR$ 16+CHR$ 9; ...

è equivalente a

PRINT INK 9; ...

Difficilmente vi preoccuperete di usare questi caratteri di controllo, dato che avete a disposizione dei comandi più comodi che svolgono la stessa funzione. Comunque possono essere utili per introdurli nei programmi, per produrre listati in differenti colori, per distinguere meglio parti differenti, oppure per avere un aspetto migliore. Ricordatevi di inserirli dopo un numero di linea, o andranno persi. Per inserire questi caratteri di controllo in un programma, dovrete scriverli sulla tastiera, usando principalmente il modo esteso e le cifre.

Nel modo esteso una cifra da 0 a 7 setta il colore della carta, e la stessa cifra con **CAPS SHIFT** setta il colore dell'inchiostro. Più precisamente, se premete il 6 per ottenere il giallo (potete usare qualunque tasto tra 0 e 7, ma non 8 e 9), vengono inseriti 2 caratteri, prima **CHR$7**, per **PAPER**, e poi **CHR$ 6**, che significa giallo. Se premete contemporaneamente **CAPS SHIFT**, viene inserito il carattere **CHR$ 16**, invece di **CHR$ 17**, per settare il colore dell'inchiostro.

Quando volete cancellare questi due caratteri, ricordatevi di premere **DELETE** due volte. Dopo che avrete premuto **DELETE** una sola volta, potranno succedere delle cose strane, vedrete apparire un punto di domanda o altro; non preoccupatevi, e premete **DELETE** una seconda volta.

⇦ e ⇨ possono comportarsi stranamente quando il cursore si muove sopra i caratteri di controllo.

Sempre nel modo esteso:

8 dà **CHR$ 19** e **CHR$ 0** per luminosità normale
9 dà **CHR$ 12** e **CHR$ 1** per la extra luminosità
CAPS SHIFT con 8 dà **CHR$ 18** e **CHR$ 0** per no lampeggiamento
CAPS SHIFT con 9 dà **CHR$ 19** e **CHR$ 1** per lampeggiamento

Ve ne sono un altro paio in modo normale:

CAPS SHIFT con 3 dà **CHR$ 20** e **CHR$ 0** per i caratteri normali
CAPS SHIFT con 4 dà **CHR$ 20** e **CHR$ 1** per i caratteri in campo inverso

Per riassumere, ecco una descrizione completa della prima fila di tasti sulla tastiera:

MODE	SHIFT	1	2	3	4	5	6	7	8	9	Ø
E	SYMBOL	**DEF FN**	**FN**	**LINE**	**OPEN #**	**CLOSE #**	**MOVE**	**ERASE**	**POINT**	**CAT**	**FORMAT**
	CAPS	Ink blue	Ink red	Ink magenta	Ink green	Ink cyan	Ink yellow	Ink white	Flash off	Flash on	Ink black
	NONE	Paper blue	Paper red	Paper magenta	Paper green	Paper cyan	Paper yellow	Paper white	Normal brightness	Extra bright	Paper black
G	EITHER	▙	▟	▄	▛	▌	▞	▜	█	EXIT GRAPHICS	**DELETE**
	NONE	▝	▘	▀	▗	▐	▚	▖	□	EXIT GRAPHICS	**DELETE**
K, L o C	CAPS	**EDIT**	CAPS LOCK	TRUE VIDEO	INVERSE VIDEO	⇦	⇩	⇧	⇨	GRAPHICS MODE	**DELETE**
	SYMBOL	!	@	#	$	%	&	'	(	)	_
	NONE	1	2	3	4	5	6	7	8	9	Ø

La funzione **ATTR** ha la forma:

ATTR (linea,colonna)

I suoi due argomenti indicano la linea e la colonna della posizione di stampa esattamente come in un comando **AT**, ma il suo valore è un numero che indica gli attributi di quella posizione. **ATTR**, come ogni altra funzione, può essere usata liberamente nelle espressioni, ecc.

Il numero ritornato da **ATTR** è il risultato della somma di quattro numeri, secondo lo schema seguente:
128 se la posizione è lampeggiante, 0 se è stabile
64 se la posizione è extra luminosa, 0 se è normale
8* per il codice del colore della carta
il codice dell'inchiostro

Per esempio, se la posizione è lampeggiante, a posizione normale, con carta gialla e inchiostro blu, i 4 numeri da sommare saranno, secondo le regole, 128, 0, 8*6=48 e 1, e fanno 177. Verificatelo con

PRINT AT 0,0; FLASH 1; PAPER 6; INK 1;" "; ATTR (0,0)

Esercizi

1. Provate

PRINT "B"; CHR$ 8; OVER 1; "/";

Dove la / ha tagliato la **B**, ha lasciato un punto bianco. La sovraimpressione dello ZX Spectrum lavora in questo modo: due carte o due inchiostri danno la carta, una carta o un inchiostro dà un inchiostro; per questa interessante proprietà, se voi sovrapponete la stessa cosa due volte, riottenete il carattere di partenza. Provate a scrivere ora:

PRINT CHR$ 8; OVER 1; "/"

perchè riottenete la **B?**

2. Scrivete

PAPER 0: INK 0

non è magnifico che questi comandi non influenzino la parte bassa dello schermo?
Scrivete ora:

BORDER 0

e osservate con quanta cura il calcolatore vi impedisce di mettervi in una situazione completamente cieca!

Fate girare il seguente programma:

```
10 POKE 22527+RND*704, RND*127
20 GO TO 10
```

Non preoccupatevi di come funziona; vi basti sapere che cambia i colori dei quadratini dello schermo, e la funzione **RND** cura che lo faccia a caso. Le strisce diagonali che osserverete sono dovute alla nascosta ricorsività dei numeri in **RND**, ricorsività che fa **RND** pseudo casuale invece di casuale.

4. Abbiate in memoria il programma per i pezzi degli scacchi del capitolo 23, fatelo girare, e introducete questo programma che disegna una scacchiera, usandoli.

```
  5 REM disegna la scacchiera vuota
 10 LET bb=1: LET bw=2: REM rosso e blu per la scacchiera
 15 PAPER bw: INK bb: CLS
 20 PLOT 79,128: REM bordo
 30 DRAW 65,0: DRAW 0,−65
 40 DRAW −65,0: DRAW 0,65
 50 PAPER bb
 60 REM bordo
 70 FOR n=0 TO 3: FOR m=0 TO 3
 80 PRINT AT 6+2*n, 11+2*m;" "
 90 PRINT AT 7+2*n, 10+2*m;" "
100 NEXT m: NEXT n
110 PAPER 8
120 LET pw=7: LET pb=ø REM colori dei pezzi bianchi e neri
200 DIM b$(8,8): REM posizioni dei pezzi
205 REM fissa posizioni iniziali
210 LET b$(1)="tcaqkact"
220 LET b$(2)="pppppppp"
230 LET b$(7)="PPPPPPPP"
240 LET b$(8)="TCAQKACT"
300 REM scacchiera video
310 FOR n=1 TO 8: FOR m=1 TO 8
320 LET bc=CODE b$(n,m): INK pw
325 IF bc=CODE" "THEN GO TO 350: REM spazio
330 IF bc>CODE "Z" THEN INK pb: LET bc=bc−32: REM casella
    più bassa dei neri
340 LET bc=bc+79: REM conversione a grafica
350 PRINT AT 5+n, 9+m; CHR$ bc
360 NEXT m: NEXT n
400 PAPER 7: INK 0
```

CAPITOLO 26

Grafica

Sommario
PLOT, DRAW, CIRCLE
POINT
pixels

In questo capitolo imparerete a disegnare sullo schermo dello ZX Spectrum. La parte di schermo che potete usare ha 22 linee e 32 colonne, che formano in tutto 22*32=704 caratteri. Come avete già visto nel capitolo 24, ogni posizione per un carattere è data da un quadrato di 8*8 punti, e ognuno di questi punti è chiamato pixel (dall'inglese picture element). Ogni pixel è individuato da due numeri, le sue *coordinate*. Il primo, la coordinata delle x, indica quanto sia lontano dalla colonna all'estrema sinistra (ricordatevi che x è orizzontale); il secondo, la coordinata delle y, indica quanto sia lontano dalla linea di base, ovvero in alto. Queste coordinate si indicano di solito con una coppia di numeri tra parentesi, tipo (0,0), (255,0), (0,175) e (255,175) sono rispettivamente l'angolo in basso a sinistra, in basso a destra, in alto a sinistra ed in alto a destra.

Il comando

PLOT coordinata x, coordinata y

colora del colore dell'inchiostro il pixel associato alle coordinate. Così il semplice programma

```
10 PLOT INT (RND*256), INT (RND*176): INPUT a$: GO TO 10
```

disegna un punto a caso ogni volta che premete **ENTER**.

Ecco un programma decisamente più interessante che disegna il grafico della funzione seno (un'onda sinusoidale), per i valori tra 0 e 2π.

```
10 FOR n=0 TO 255
20 PLOT n,88+80*SIN (n/128*PI)
30 NEXT n
```

Il prossimo programma disegna un grafico di **SQR**, (un ramo di una parabola), tra 0 e 4.

```
10 FOR n=0 TO 255
20 PLOT n,80*SQR (n/64)
30 NEXT n
```

Notate che le coordinate dei pixels sono piuttosto differenti dai numeri che indicano le linee e le colonne del comando **AT**. Osservate il diagramma del capitolo 24 e tenetelo sottomano quando lavorate con le coordinate e i numeri di linea e di colonna.

Per aiutarvi a disegnare, il calcolatore è in grado di disegnare da solo linee diritte, cerchi e parti di cerchi, usando i comandi **DRAW** e **CIRCLE**.

Il comando **DRAW** disegna un segmento di linea retta, e si usa con la sintassi

```
DRAW x,y
```

La linea disegnata inizia dalla posizione di **PLOT**, ovvero da dove l'ultimo comando grafico eseguito ha lasciato detta posizione (**RUN, CLEAR, CLS** e **NEW** la resettano all'angolo sinistro in basso (0,0)) e termina al pixel determinato dalle coordinate x e y, intese come distanze relative alla posizione di partenza. Per disegnare una linea tra due punti desiderati, eseguite prima una **PLOT** per portarvi al punto di partenza, e quindi una **DRAW** con le distanze del punto di arrivo della linea dal primo.

Provate qualche comando, per esempio:

```
PLOT 0,100: DRAW 80,-35
PLOT 90,150: DRAW 80,-35
```

Notate che un numero in un comando **DRAW** può anche essere negativo, diversamente dal comando **PLOT**.

Se volete, potete anche eseguire **PLOT** e **DRAW** a colori, per quanto dobbiate sempre tenere ben presente che i colori riguardano sempre un'intera posizione di carattere, e non possono essere specificati per un singolo pixel. Un pixel viene disegnato usando il colore corrente, e tutta la posizione di carettere che lo contiene assume questo valore. Convincetevene facendo girare questo programma:

```
10 BORDER 0: PAPER 0: INK 7: CLS : REM nero fuori dallo schermo
20 LET x1=0: LET y1=0: REM inizio della linea
30 LET c=1: REM inizio del colore blu per l'inchiostro
40 LET x2=INT (RND*256): LET y2=INT (RND*176):REM
   fine casuale della linea
50 DRAW INK c;x2-x1,y2-y1
60 LET x1=x2: LET y1=y2: REM la prossima linea inizia dove è finita
   l'ultima
70 LET c=c+1: IF c=8 THEN LET c=1: REM colore nuovo
80 GO TO 40
```

La linea sembra diventare più larga man mano che il programma continua, dato che cambia il colore di tutti i pixels mostrati in ogni posizione di carattere che attraversa. Notate che **PAPER, INK, FLASH, BRIGHT, INVERSE** e **OVER** posso-

no essere introdotti come elementi in un comando **PLOT** o **DRAW**, esattamente come in un comando **PRINT** o **INPUT**. Vanno introdotti tra la parola chiave e le coordinate, e devono essere seguiti da virgola, oppure da punto e virgola.

DRAW permette anche di disegnare parti di circonferenze, invece che linee diritte, usando un terzo numero per specificare l'angolo di ampiezza della circonferenza. La sintassi è

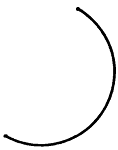
DRAW x,y,a

x e **y** sono usati per specificare il punto di arrivo della linea, esattamente come prima, e **a** è il numero di radianti dell'angolo corrispondente alla circonferenza: **a** positivo produce una circonferenza rivolta verso sinistra, **a** negativo rivolta verso destra; **a** può essere anche vista come la frazione di cerchio che si vuole far disegnare, tenendo presente che un cerchio è composto da 2π radianti. Quindi a=π disegnerà un semicerchio, a=0.5π un quarto di cerchio, e così via.

Per esempio, se **a**=π, qualunque valore assegnato a **x** e a **y**, sarà disegnato un semicerchio. Fate girare

10 PLOT 100,100: DRAW 50,50, PI

e disegnerà



Il disegno inizia andando in direzione sud-est, ma termina verso nord-ovest. Durante il suo percorso ha girato di 180 gradi (π radianti, il valore di **a**). Fate girare il programma diverse volte, sostituendo a **PI** espressioni a piacere, per esempio **−PI, PI/2, 3*PI/2, PI/4, 1,0.**

L'ultimo comando esaminato in questo capitolo è **CIRCLE**, che disegna una circonferenza completa. Specificate le coordinate del centro, il raggio del cerchio con la sintassi:

CIRCLE coordinata x,coordinata y, raggio

Esattamente come per **PLOT** e **DRAW, CIRCLE** accetta tutti i vari comandi di colore.

La funzione **POINT** vi dice se un pixel è inchiostrato o no. Ha due argomenti, cioè le coordinate del pixel, che devono essere racchiuse tra parentesi, e ritorna 0 se il pixel è del colore della carta, 1 se dell'inchiostro. Provate

CLS: PRINT POINT (0,0): PLOT 0,0: PRINT POINT (0,0)

Scrivete

PAPER 7, INK 0

e cercate di capire come **INVERSE** e **OVER** lavorano con la **PLOT**. Queste due influenzano solo il pixel disegnato, e non la rimanente parte di posizione del carattere. Dato che sono normalmente disattivate (0), per attivarle (1) dovrete includerle in un comando di **PLOT**.

Ecco una lista dei possibili comandi da tenere presenti:
PLOT; - è nella forma normale. Esso inchiostra un punto, ovvero fa si che il pixel assuma il colore dell'inchiostro.
PLOT INVERSE 1; - colora un punto con la scolorina, ovvero fa si che il pixel assuma il colore della carta.
PLOT OVER 1; - inverte il colore del pixel. Se prima era del colore della carta, lo fa diventare del colore dell'inchiostro, e viceversa.
PLOT INVERSE 1; OVER 1; - lascia il pixel esattamente invariato, ma, dato che sposta la posizione di stampa, troverete comodo usarlo.

Per avere un altro esempio dell'uso di **OVER**, riempite la schermo con scritte nere su sfondo bianco e scrivete:

PLOT 0,0: DRAW OVER 1;275,175

Otterrete una linea abbastanza decente, anche se presenta delle interruzioni ogni volta che attraversa delle scritte. Ora rieseguite esattamente lo stesso comando. La linea sparisce senza lasciare alcuna traccia. Il grande vantaggio di **OVER** è che vi permette di disegnare e di cancellare senza rovinare quello che c'è sullo schermo. Se aveste disegnato la linea usando

PLOT 0,0: DRAW 255,175

e poi l'aveste cancellata usando

PLOT 0,0: DRAW INVERSE 1;255,175

avreste cancellato anche parte delle scritte.

Provate adesso

PLOT 0,0: DRAW OVER 1;250,175

e provate a cancellarla con

DRAW OVER 1;−250,−175

Non funzionerà, dato che i pixels usati ritornando non sono esattamente gli stessi usati andando in giù. Quando cancellate una linea, la dovete cancellare esattamente nella stessa direzione in cui l'avete tracciata.

Un modo per ottenere colori strani è di mescolarne due insieme in un solo quadrato, usando un carattere grafico definito dall'utente. Fate girare questo programma:

```
1000 FOR n=0 TO 6 STEP 2
1010 POKE USR "a"+n,BIN 01010101: POKE USR "a"+n+1,
     BIN 10101010
1020 NEXT n
```

che definisce un carattere grafico equivalente a una scacchiera. Se stampate questo carattere (in modo grafico è **a**), in inchiostro rosso su carta gialla e viceversa, vi accorgerete che dà un arancione di qualità discreta.

Esercizi

1. Divertitevi con **PAPER, INK, FLASH** e **BRIGHT** nei comandi **PLOT**. Ricordatevi che influiscono su tutta la posizione di carattere che contiene il pixel. Normalmente è come se **PLOT** fosse nella forma

```
PLOT PAPER 8; FLASH 8; BRIGHT 8; ...
```

solo il colore dell'inchiostro della posizione di carattere si è alterato quando qualcosa vi viene disegnato, ma potete cambiarlo a vostro piacimento.

Prestate particolare attenzione quando usate **INVERSE 1**, dato che colora un pixel col colore della carta, ma cambia il colore dell'inchiostro, e può darsi che non sia l'effetto che volevate ottenere.

2. Provate a disegnare cerchi usando **SIN** e **COS** (se avete già letto il capitolo 18, cercate di trovare da soli come). Fate girare questo

```
10 FOR n=0 TO 2*PI STEP PI/180
20 PLOT 100+80*COS n,87+80*SIN n
30 NEXT n
40 CIRCLE 150,87,80
```

Notate come il comando **CIRLE** sia molto più veloce, anche se meno preciso.

3. Provate

```
CIRCLE 100,87,80: DRAW 50,50
```

Potete osservare che il comando **CIRCLE** lascia la posizione di **PLOT** in un punto abbastanza indeterminato, da qualche parte, a mezza strada, verso il lato destro del cerchio. È una buona pratica usare una **PLOT** dopo una **CIRCLE** prima di fare altri disegni.

4. Segue un programma per disegnare il grafico di quasi ogni funzione. Prima chiede un numero **n**, che userà per stampare un grafico da −**n** a +**n** quindi chiede la funzione, letta come una stringa, che dovrà essere un'espressione, usando **x** come argomento della funzione.

```
 10 PLOT 0,87: DRAW 255,0
 20 PLOT 127,0: DRAW 0,175
 30 INPUT s,e$
 35 LET t=ø
 40 FOR f=0 TO 255
 50 LET x=(f—128)*s/128: LET y=VAL e$
 60 IF ABS y>87 THEN LET t=0: GO TO 100
 70 IF NOT t THEN PLOT f,y+88: LET t=1: GO TO 100
 80 DRAW 1,y—old y
100 LET old y=INT (y+.5)
110 NEXT f
```

Fatelo girare, e, per esempio, introducete **10** come numero **n** e **10*TAN x** come funzione. Stamperà il grafico della tangente di *x*, con *x* che varia da −10 a +10.

CAPITOLO 27

Movimento

Sommario
PAUSE, INKEY$, PEEK

Talvolta può rendersi necessaria una pausa ben definita, durante la quale il programma non svolga alcuna operazione: in questo caso troverete utile il comando **PAUSE**.

PAUSE n

arresta l'esecuzione del programma, mantenendo attivo il video per *n* scansioni televisive (in un secondo vi sono 50 scansioni); *n* può essere al massimo 65535, il che ferma il programma per poco meno di 22 minuti se invece *n*=0, il programma si arresterà per sempre.

Una pausa prodotta in questo modo può sempre essere accorciata premendo un qualunque tasto (ma, come al solito, lo spazio con **CAPS SHIFT** produce un **BREAK**). Ovviamente, per accorciare una pausa, il tasto dovrà essere premuto dopo l'inizio della stessa.

Il seguente programma simula la lancetta dei secondi di un orologio:

```
10 REM Disegno del quadrante dell'orologio
20 FOR n=1 TO 12
30 PRINT AT 10-10*COS (n/6*PI), 16+10*SIN (n/6*PI);n
40 NEXT n
50 REM Simulazione della lancetta dei secondi
60 FOR t=0 TO 200000: REM t è il tempo in secondi
70 LET a=t/30*PI : REM a è l'angolo della lancetta dei secondi espresso
   in radianti
80 LET sx=80*SIN a: LET sy=80*COS a
200 PLOT 128,88: DRAW OVER 1;sx,sy:REM disegna la lancetta dei secondi
210 PAUSE 42
220 PLOT 128,88: DRAW OVER 1;sx,sy: REM cancella la lancetta dei secondi
400 NEXT t
```

Questo orologio si ferma dopo 55 ore e mezzo circa, a meno che voi non cambiate la linea 60. Notate come si sia ottenuta la precisione dell'orologio. Vi sareste potuti aspettare **PAUSE 50** alla linea 210, per far passare un secondo, ma, dato che il calcolatore impiega qualche tempo per eseguire le linee del ciclo **FOR-NEXT** la linea 210 ferma il calcolatore solo per il tempo che rimane. Il modo migliore per tarare un orologio di questo tipo è di procedere per tentativi, facendo una prima

stima della *n*, e correggendola fino ad ottenere una reale simulazione di un orologio vero. Comunque non è possibile ottenere una grande precisione, dato che la minima correzione applicabile è di una scansione per un secondo, equivalente al 2%, cioè mezzora al giorno.

Qualora fosse necessario misurare il tempo con maggior precisione, è possibile farlo leggendo il contenuto di certe locazioni di memoria, usando la **PEEK**. Il capitolo 34 spiega quanto segue più in dettaglio. L'espressione usata è

(65536*PEEK 23674+256*PEEK 23673+PEEK 23672)/50

Ritorna il numero di secondi trascorsi dall'accensione del calcolatore, fin a circa tre giorni e 21 ore, dopo di che si resetta.

Ecco il programma dell'orologio, modificato per usare la formula più precisa.

```
 10 REM Disegno del quadrante dell'orologio
 20 FOR n=1 TO 12
 30 PRINT AT 10-10*COS (n/6*PI), 16+10*SIN (n/6*PI);n
 40 NEXT n
 50 DEF FN t()=INT((65536*PEEK 23674+256*PEEK 23673+PEEK
    23672)/50): REM numero di secondi dall'inizio
100 REM simulazione della lancetta dei secondi
110 LET t1=FN t()
120 LET a=t1/30*PI: REM a è l'angolo della lancetta dei secondi espresso
    in radianti
130 LET sx=72* SIN a: LET sy=72* COS a
140 PLOT 131,91: DRAW OVER 1;sx,sy: REM disegna la lancetta
200 LET t=FN t()
210 IF t<=t1 THEN GO TO 200: REM attende fino a quando è ora
    di muovere la lancetta
220 PLOT 131,91: DRAW OVER 1;sx,sy: REM cancella la vecchia lancetta.
230 LET t1=t: GO TO 120
```

L'orologio interno usato ha una precisione di circa lo 0.01%, cioè dieci secondi al giorno, fino a quando il programma gira, ma si ferma momentaneamente ogni volta che viene eseguito **BEEP**, o un'operazione su cassetta, o con la stampante, o con qualunque altra periferica usata dal calcolatore. L'orologio rimarrà indietro nel tempo impiegato da tutte queste operazioni.

I numeri **PEEK 23674, PEEK 23673** e **PEEK 23672** sono usati all'interno del calcolatore per contare in cinquantesimi di secondo. Ognuno varia da 0 a 255, e una volta raggiunto 255, ricomincia da 0.

PEEK 23672 aumenta più rapidamente di tutti, incrementandosi di 1 ogni cinquantesimo di secondo. Quando, dopo aver raggiunto 255, torna a 0, incrementa di 1 **PEEK 23673**, e quando quest'ultimo, ogni 250/50 secondi, raggiunge i 255 e torna a 0, incrementa di 1 **PEEK 23674**. Adesso non vi sarà difficile capire come funziona il programma dell'orologio.

Prestate molta attenzione all'esempio seguente: supponete che i tre numeri siano 0 (per **PEEK 23674**), 255 (per **PEEK 23673**) e 255 (per **PEEK 23672**), siano trascorsi cioè circa 21 minuti dall'accensione. L'espressione diventa

(65536*0+256*255+255)/50=1310,7

Vi è un pericolo nascosto. Tra un cinquantesimo di secondo i tre numeri cambieranno in 1, 0 e 0. Inevitabilmente, ogni tanto, questo capiterà proprio mentre state calcolando l'espressione, facendo si che il calcolatore legga **PEEK 23674** come 0, e cambiando gli altri due in 0 prima che li possa leggere. L'espressione diventerebbe quindi

(65536*0+256*0+0)/50=0

che è completamente errato.

Un modo facile di evitare questo problema è di calcolare due volte l'espressione in sequenza, e prendere il valore più grosso. Guardate con attenzione il programma dell'orologio e scoprirete che lo fa implicitamente. Un trucco per applicare la regola è di definire la funzione:

```
10 DEF FN m(x,y)=(x+y+ABS (x-y))/2: REM il maggiore tra x e y
20 DEF FN u()=(65536*PEEK 23674+256*PEEK 23673+PEEK 23672)/50:
   REM tempo senza garanzie di esattezza
30 DEF FN t()=FN m(FN u()): REM tempo esatto
```

Per ottenere l'ora del giorno invece del numero di secondi dall'accensione del calcolatore, è sufficiente scrivere nelle tre locazioni di memoria l'orario convertito in secondi.

Per esempio, per posizionare l'orologio sulle 10, si calcola che 10*60*60*50=1800000 cinquantesimi di secondo, e quindi che

1800000=65536*27+256*119+64

e si memorizzano i numeri 27, 119 e 64 con

POKE 23674,27: POKE 23673,119: POKE 23672,64

Nei paesi con la frequenza di rete di 60 Hertz, è necessario cambiare, negli opportuni punti dei programmi, 50 in 60.

La funzione **INKEY$**, che è priva di argomento, ritorna il carattere premuto sulla tastiera al momento della sua chiamata. Quando non è premuto alcun tasto, la funzione **INKEY$** ha valore di stringa nulla.

Con questa funzione è facile trasformare il calcolatore in una macchina da scrivere, con un programma come il seguente:

```
10 IF INKEY$ < > “” THEN GO TO 10
20 IF INKEY$=“” THEN GO TO 20
30 PRINT INKEY$;
40 GO TO 10
```

La linea 10 attende che lasciate l'ultimo tasto premuto, e la linea 20 aspetta che ne premiate uno nuovo. Ricordate che, diversamente da **INPUT, INKEY$** non aspetta ne che premiate **ENTER**, ne che premiate alcun tasto. Con **INKEY$** non si usa **ENTER**, ma se non viene premuto alcun tasto, ritorna il valore di stringa nulla e il programma continua.

Esercizi

1. Cercate di prevedere cosa succede se togliete la linea 10 dal programma della macchina da scrivere: provate per verificare le vostre congetture.

2. **INKEY$** può essere usato insieme a **PAUSE**, ad esempio in questa versione del programma della macchina da scrivere:

```
10 PAUSE 0
20 PRINT INKEY$;
30 GO TO 10
```

Perchè, per il funzionamento di questo programma, è indispensabile che la pausa non finisca, se trova un tasto già premuto quando inizia?

3. Modificate il programma dell'orologio in modo che mostri anche la lancetta dei minuti e delle ore, ridisegnandole ogni minuto. Per renderlo più sofisticato, fate poi in modo che, ogni quarto d'ora, faccia qualcosa di appariscente, per esempio imitando i rintocchi del famoso orologio di Londra, il Big Ben, col comando **BEEP**, spiegato nel prossimo capitolo.

4. (Per sadici). Provate:

```
10 IF INKEY$=“” THEN GO TO 10
20 PRINT AT 11,14;“ahi!”
30 IF INKEY$< >“” THEN GO TO 30
40 PRINT AT 11,14; “     ”: REM lascia solo il!
50 GO TO 10
```

CAPITOLO 28

Produzione di suoni

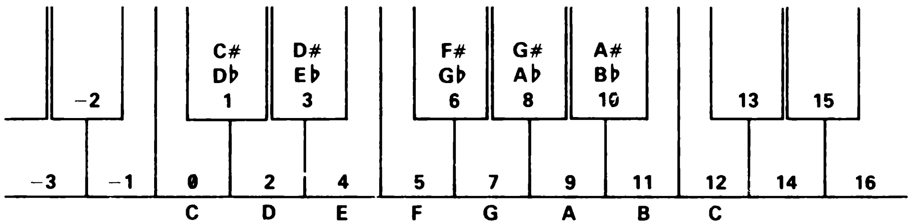
Sommario
BEEP

Se non avete ancora scoperto che lo ZX Spectrum ha un altoparlante incorporato, leggete i capitoli introduttivi prima di andare avanti. Il comando

BEEP durata, altezza

fa sì che l'altoparlante emetta un suono della durata e dell'altezza specificata, dove durata e altezza sono una qualunque espressione numerica. La durata è espressa in secondi, e l'altezza in semitoni sopra il DO centrale, con numeri negativi per le note sotto quest'ultimo.

Segue un diagramma che indica i valori di altezza per le note di un ottava di pianoforte.



ricordate che C=DO, D=RE, E=MI, F=FA, G=SOL, A=LA e B=SI.

Per produrre note in ottave superiori o inferiori, dovete sommare o sottrarre 12 per ogni ottava di differenza.

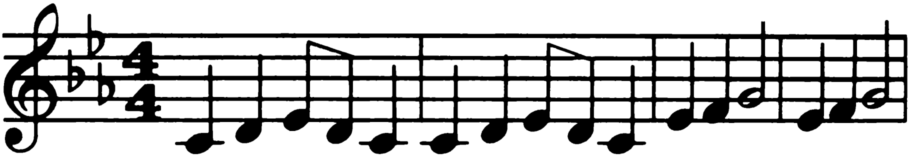
Se memorizzate una melodia che sapete suonare al pianoforte, basterà dare un'occhiata al grafico per sapere i valori di tutte le altezze delle note. Se trascrivete da musica scritta, vi suggeriamo di scrivere di fianco ad ogni spazio ed ogni linea del pentagramma il valore dell'altezza corrispondente, tenendo in considerazione l'armatura di chiave.

Per esempio, battete

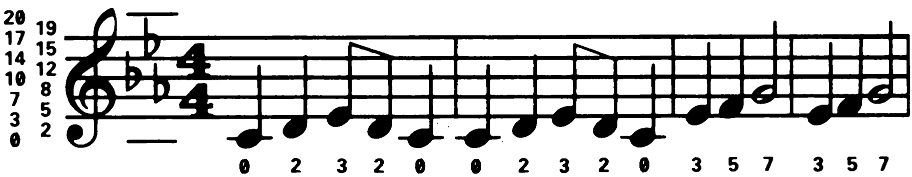
```
10 PRINT "Frere Gustav"
20 BEEP 1,0: BEEP 1,2: BEEP .5,3: BEEP .5,2: BEEP 1,0
30 BEEP 1,0: BEEP 1,2: BEEP .5,3: BEEP .5,2: BEEP 1,0
40 BEEP 1,3: BEEP 1,5: BEEP 2,7
50 BEEP 1,3: BEEP 1,5: BEEP 2,7
60 BEEP .75,7: BEEP .25,8: BEEP .5,7: BEEP .5,5: BEEP .5,3: BEEP .5,2:
   BEEP 1,0
70 BEEP .75,7: BEEP .25,8: BEEP .5,7: BEEP .5,5: BEEP .5,3: BEEP .5,2:
   BEEP 1,0
80 BEEP 1,0: BEEP 1,—5: BEEP 2,0
90 BEEP 1,0: BEEP 1,—5: BEEP 2,0
```

Fate girare il programma per sentire la marcia funebre dalla prima sinfonia di Mahler.

Supponete, per esempio, che la melodia che volete trascrivere sia scritta nella chiave di DO minore, come quella di Mahler, il cui inizio è



e scrivete i valori delle altezze delle note così:



Abbiamo aggiunto la prima sopra e sotto linea per riferimento. Se non siete dotti in musica, osservate che il MI bemolle nell'armatura di chiave, influisce non solo sul MI nello spazio più alto, abbassandolo da 16 a 15, ma anche sul MI nella riga più bassa, abbassandolo da 4 a 3. Con un minimo di allenamento il lavoro di trascrizione sarà fatto comodamente a mente.

Se volete cambiare la chiave del pezzo, dovete sommare al valore dell'altezza di ogni nota una variabile (per esempio **chiave**), cui dovrà essere attribuito l'appropriato valore prima dell'esecuzione del pezzo. La linea 20 del programma diventerebbe

```
20 BEEP 1,chiave+0: BEEP 1,chiave+2: BEEP .5,chiave+3:
   BEEP .5,chiave+2: BEEP 1,chiave+0
```

Sempre nel nostro esempio, la variabile **chiave** dovrebbe avere il valore 0 per DO minore, 2 per RE minore e 12 sempre per DO minore, ma nell'ottava più alta e così via. Con questo sistema è anche possibile accordare il calcolatore con un altro strumento, usando valori decimali per la variabile **chiave**. Analogamente è anche possibile avere un pezzo eseguibile a diverse velocità, nell' esempio, dato che si tratta di un pezzo abbastanza lento, si è fatto durare ogni quarto un secondo, e le altre note di conseguenza, mezzo, un quarto di secondo, due secondi, ecc.

Se si introduce una variabile **quarto** analogamente alla variabile chiave, la linea 20 diventa

```
20 BEEP quarto, chiave+0: BEEP quarto,chiave+2: BEEP quarto/
   2,chiave+3: BEEP quarto/2,chiave+2: BEEP quarto,chiave+0
```

Rendendo possibile l'esecuzione dello stesso programma in qualunque chiave a qualunque velocità, e, ovviamente, con qualunque accordatura. Tenete però sempre presente che il calcolatore non è in grado di suonare più di una nota alla volta; potrete quindi rendere completamente solo melodie non armonizzate.

Provate a programmare qualche canzone da soli, cominciando dalle più semplici. Se non avete ne un pianoforte, ne musica scritta, un qualunque semplice strumento (magari un flauto dolce), o al limite della musica registrata, saranno adatti per cercare una melodia. Per procedere agevolmente nel lavoro è opportuno che facciate un diagramma con i valori delle altezze e tutte le note che potete suonare sul vostro strumento, a meno che non riusciate a tenerlo a mente. Scrivete

```
FOR n=0 TO 1000: BEEP .5,n: NEXT n
```

Suonerà le note sempre più acute fino al limite delle possibilità del calcolatore, raggiunte le quali stamperà il messaggio **B integer out of range**. Stampate quindi **n** per scoprire la nota più acuta che potete suonare.

Ripetete lo stesso procedimento, ma andando verso il basso. Le note più basse suoneranno piuttosto come dei clicks, invece che come note, ma gli stessi clicks, ripetuti velocemente producono una nota distinguibile dall'orecchio.

La gamma media è la più adatta per suonare delle musiche; le note basse suonano troppo come clicks, e le note alte sono troppo deboli ed imprecise.

Scrivete questa linea di programma:

```
10 BEEP .5,0: BEEP .5,2: BEEP .5,4: BEEP .5,5: BEEP .5,7:
   BEEP .7,9: BEEP .5,11: BEEP .5,12: STOP
```

Suona la scala temperata di DO maggiore, che usa tutte le note bianche del pianoforte, dal DO centrale al DO superiore. Questa scala è accordata esattamente alla stessa maniera del pianoforte, ed è chiamata scala temperata, dato che usa lo stesso intervallo di semitono su tutta la scala. Un violista invece suonerebbe la scala appena un poco differentemente, spostando di poco tutte le note per farle suonare più piacevoli all'orecchio. Sul violino, differentemente che su una tastiera di pianoforte, si può muovere un dito un pò in su o in giù a piacere. La scala naturale suonata da un violinista basata solo sul fenomeno dei suoni armonici risulterebbe così:

```
20 BEEP .5,0: BEEP .5,2.039: BEEP .5,3.86: BEEP .5,4.98:
   BEEP .5,7.02: BEEP .5,8.84: BEEP .5,10.88: BEEP .5,88:STOP
```

Sentitele una dopo l'altra: può darsi che non siate in grado di distinguerle; alcune persone ci riescono. La prima differenza apprezzabile è che la terza nota è legger-

mente più bassa sulla scala naturale che sulla scala temperata. Se siete un vero perfezionista, potrete programmare le vostre melodie su questa scala naturale. Lo svantaggio è, che per quanto funzioni perfettamente in chiave di DO, non lavora altrettanto bene su tutte le altre chiavi, e anzi, lavora molto male in alcune, dato che ogni chiave ha la sua propria scala naturale. La scala temperata è appena un pelo diversa, e lavora ugualmente bene su tutte le chiavi.

Questo comunque non è un problema su un calcolatore; usando un trucco come quello della variabile **chiave**.

Qualche musica, come la musica indiana per esempio, usa intervalli più brevi di un semitono. Fortunatamente, per il calcolatore neanche questi rappresentano un problema; un **BEEP** sopra il DO centrale di un quarto di tono ha un valore di altezza 0.5. Se volete far sì che la tastiera suoni invece di emettere un click, eseguite

POKE 23609,255

Il secondo numero del comando determina la lunghezza del BEEP, provate diversi valori tra 0 e 255. Se è 0 il BEEP sarà così corto da sembrare un click.

Se non vi basta la qualità ed il volume resi dall'altoparlante incorporato nello Spectrum, potete collegarvi ad una cuffia o perfino ad un amplificatore, dato che il segnale è presente su entrambe le prese sul registratore, "MIC" e "EAR". MIC dà un'uscita a livello più basso, adatta al collegamento con un amplificatore HI-FI; EAR invece ha un livello di uscita più alto, da collegare ad un auricolare o una cuffietta. Escludendo i livelli, i due segnali sono perfettamente identici, e comunque l'uso di apparecchi esterni non disabilita l'altoparlante incorporato. Non abbiate paura di provare a eseguire collegamenti esterni, dato che anche se cortocircuitate le prese "MIC" e "EAR" non potete danneggiare lo Spectrum. Può essere interessante registrare lo Spectrum da registratore e poi farlo risuonare sopra la melodia registrata.

Esercizi

1. Riscrivete il programma di Mahler nel capitolo 17 in modo che usi i cicli **FOR** per ripetere le battute.

Programmate il calcolatore per fargli suonare non solo la marcia funebre, ma tutta la prima sinfonia di Mahler.

CAPITOLO 29

Uso avanzato del registratore a cassette

Sommario
LOAD, SAVE, VERIFY, MERGE

Prima di procedere nella lettura di questo capitolo, provate, se non l'avete ancora fatto, le procedure base per registrare, caricare e verificare i programmi (**SAVE, LOAD, VERIFY**), spiegate nel capitolo 6.

Di seguito si esaminano le caratteristiche dei comandi più avanzati. Il comando **MERGE** carica un programma registrato su cassetta nella memoria del calcolatore, ma diversamente dal comando **LOAD** non cancella, prima di iniziare il caricamento, il vecchio programma e le vecchie variabili. **MERGE** cancella, e non potrebbe fare diversamente, solo quelle linee o quelle variabili che siano presenti anche nel programma che carica da cassetta. Introducete nuovamente il programma "**dadi**" del capitolo 20, e memorizzatelo su cassetta col nome "**dadi**". Quindi introducete e fate girare il seguente:

```
 1 PRINT 1
 2 PRINT 2
10 PRINT 10
20 LET x=20
```

e quindi procedete come per la verifica della registrazione, ma usando

MERGE "dadi"

invece di

VERIFY "dadi"

Listate il programma, e verificate che le linee 1 e 2 siano rimaste, mentre le linee 10 e 20 sono state rimpiazzate dalle omonime del programma dei dadi. Eseguite **PRINT x** per verificare che anche il valore di **x** non è stato cancellato.

Con questo avete visto la forma più elementare dei 4 comandi che possono essere usati col registratore a cassette. Per riassumere:

SAVE memorizza programma e variabili su cassetta.

VERIFY verifica che il programma e le variabili registrate su cassetta siano uguali a quelle nella memoria del calcolatore.

LOAD cancella il programma e tutte le variabili, e le sostituisce con i dati memorizzati sulla cassetta.

MERGE è analogo a **LOAD**, ma non esegue l'operazione preliminare di cancellare le vecchie linee di programma e le vecchie variabili, che quindi rimangono inalterate, a meno che non si debba sostituirle con omonime del nuovo programma.

Ognuno di questi comandi è seguito da una stringa, che indica, nel caso della **SAVE**, il nome con cui memorizzare il programma, e nel caso degli altri comandi, il nome del programma da cercare. Durante la ricerca del programma specificato, il calcolatore stampa il nome di ogni programma che incontra. Se con **VERIFY, LOAD** e **MERGE** usate una stringa nulla, come nome del file da cercare, il calcolatore caricherà il primo programma che incontra, senza preoccuparsi del nome.

Quando si registra un programma su cassetta, è possibile registrarlo in modo tale che, quando viene ricaricato in memoria, venga automaticamente lanciato a partire da una certa linea.

SAVE stringa **LINE** numero

fà sì che il programma caricato con **LOAD** (ma non con **MERGE**), inizi automaticamente a girare a partire dalla linea specificata con *numero*.

Oltre ai programmi e relative variabili, si possono anche memorizzare altri due tipi di dati, chiamati uno *arrays*, o *matrici*, e l'altro **bytes**.

Per memorizzare una matrice, si usa **DATA** in un comando **SAVE** nella forma

SAVE stringa **DATA** nome di matrice ()

Stringa è ancora il nome che l'informazione avrà su nastro, esattamente come per i programmi. Il *nome di matrice* specifica la matrice che si desidera memorizzare, quindi può essere una lettera singola, per una matrice numerica, o una lettera singola, seguita da $, per una matrice di stringhe. Non dimenticatevi le parentesi finali, anche se a prima vista possono apparire logicalmente inutili. Siate certi di non confondere le funzioni di *stringa* e *nome di matrice*. Per esempio, se scrivete

SAVE "Pippo" DATA b()

la matrice numerica **b** viene prelevata dalla memoria del calcolatore e registrata su nastro col nome **Pippo**. Analogamente, quando voi scrivete

VERIFY "Pippo" DATA b()

il calcolatore cerca su cassetta una matrice memorizzata con nome **Pippo** (quando la trova scrive **Number array: Pippo**), e la confronta con la matrice **b** in memoria.

LOAD "Pippo" DATA b()

cerca la matrice su nastro e, se c'è abbastanza memoria libera, cancella un eventuale preesistente matrice **b**, e carica la nuova matrice dal nastro, chiamandola **b** a sua volta.

MERGE non può essere usato con le matrici su nastro.

Nell'esempio si è parlato di matrici numeriche, dato che **b()** è il nome di una matrice numerica. Con matrici di stringhe, tutto funziona esattamente allo stesso modo, salvo che il calcolatore scrive **Character array:**, seguito dal nome, quando trova la matrice. Ricordatevi che quando caricate una matrice stringa da nastro, cancellate non solo un'eventuale matrice stringa avente lo stesso nome, ma anche un'eventuale stringa, ovviamente sempre con lo stesso nome.

La memorizzazione tipo byte è usata per qualunque tipo di dato, senza alcun riferimento all'uso del dato stesso. Può trattarsi di un'immagine televisiva, di un carattere grafico definito dall'utente, o di qualcosa definito da voi stessi. La memorizzazione tipo byte si ottiene usando **CODE**, nel comando **SAVE**, nella forma

SAVE stringa **CODE** byte d'inizio, numero dei byte

Per poter capire come funziona questo comando è necessario fare un paio di premesse. L'unità di memoria interna del calcolatore è il *byte*; il contenuto di un byte non è altro che un numero tra 0 e 255, associato con un *indirizzo* (che a sua volta è un numero tra 0 e 65535), che lo distingue e lo ordina rispetto agli altri bytes, rendendolo unico. In questa memoria viene conservato tutto ciò che serve al calcolatore per il suo funzionamento, il programma codificato in modo particolare, le variabili, l'immagine per il video, ecc. Ognuno di questi inizia ad un certo indirizzo, e occupa un certo numero di bytes nella memoria. Col modo di memorizzazione byte si copia una parte della memoria interna del calcolatore, esattamente così com'è, su nastro. *Byte d'inizio* è l'indirizzo del primo byte che si vuole memorizzare, e *numero dei byte* è il numero dei byte che si vuole memorizzare, ovvero la lunghezza della zona di memoria che si vuole copiare. Per esempio, la zona di memoria dove è conservata l'immagine televisiva inizia all'indirizzo 16384, ed è lunga 6912 bytes; quindi col comando

SAVE "immagine" CODE 16384,6912

si copia l'immagine del video al momento dell'esecuzione del comando su nastro col nome **immagine**. È inutile dire che la stringa che segue **SAVE** è la stessa cosa che per i programmi. Eseguite il comando e, dopo aver provocato qualche modifica all'immagine dello schermo, ricaricate l'immagine con **LOAD "immagine" CODE**. Potrete così apprezzare tra l'altro come viene memorizzata l'immagine video in memoria.

È anche possibile specificare l'indirizzo da cui iniziare a memorizzare i bytes, ed eventualmente anche il numero dei bytes da caricare, nella forma

LOAD nome **CODE** indirizzo di partenza, lunghezza

In quest'ultimo caso la lunghezza è solo una misura di precauzione; infatti il calcolatore non caricherà dei bytes da cassetta (dopo aver ovviamente verificato che il nome sia giusto), se ve ne sono più che la lunghezza specificata. Una discordanza di lunghezze indica ovviamente un errore, e quindi il calcolatore evita

che si possano cancellare, sostituendoli con i dati nuovi, i contenuti di una parte di memoria che potrebbero rivelarsi utili. In caso di discordanza di lunghezze si ottiene l'errore **R Tape loading error**. Se non vi volete preoccupare ogni volta della lunghezza dei byte da caricare, potete tralasciare la specificazione della lunghezza; in questo caso il calcolatore caricherà tutti i bytes registrati su cassetta.

Indirizzo di partenza indica l'indirizzo in cui deve essere caricato il primo byte, e può anche essere diverso dall'indirizzo da cui era stato prelevato originariamente. Se si tralascia l'indirizzo di partenza nel comando di **LOAD**, i bytes caricati da cassetta tornano ad occupare la posizione originale, che avevano cioè prima di essere memorizzati.

CODE 16384,6912, è di uso così frequente che è stata predisposta un'abbreviazione **SREEN$** per facilitarne l'uso quindi i comandi per memorizzare l'immagine video sopra riportati diventano

SAVE "immagine" SCREEN$
LOAD "immagine" SCREEN$

Notate che quando memorizzate l'immagine video non potete usare la **VERIFY**; infatti **VERIFY** stampa il nome di quello che trova sul nastro, modificando così l'immagine video, che non risulterà più uguale a quella memorizzata, rendendo quindi impossibile la verifica. Ricordatevi comunque che è prudente usare sempre **VERIFY** dopo **SAVE** in tutti gli altri casi, dato che prima o poi vi capiterà di eseguire male una registrazione, rischiando di perdere qualcosa di utile.

Segue un quadro riassuntivo di tutti i comandi utilizzabili dal registratore a cassette. Per nome si intende qualunque espressione stringa, e si indica il nome con cui l'informazione è memorizzata sulla cassetta. Deve essere formato solo da caratteri ASCII stampabili, e comunque non ne vengono usati che i primi 10.

Vi sono 4 tipi di informazioni che possono essere memorizzati su nastro: i programmi insieme alle loro variabili, le matrici numeriche, le matrici stringa e i bytes semplici.

Quando **VERIFY, LOAD** o **MERGE** cercano sul nastro un'informazione di un certo tipo e dato nome, stampano sullo schermo i tipi e i nomi di tutte le informazioni che incontrano. Il tipo è indicato da **Program**, o **Number array**, o **Character array**, o **Bytes**, seguiti dal nome se quest'ultimo è stato specificato; se essa è una stringa vuota, allora caricano la prima informazione del tipo cercato, senza considerarne il nome.

SAVE
Memorizza informazioni su nastro col nome dato. Se un nome è formato da una stringa nulla, o più di 10 caratteri, si ha un errore di tipo F.

SAVE stampa sempre il messaggio **Start tape, then press any key** (avviate il nastro e poi premete un tasto), e attende quindi che sia premuto un tasto prima di iniziare a memorizzare.

1. programmi e variabili:

SAVE nome

è la forma normale.

SAVE nome **LINE** numero di linea

memorizza il programma e le variabili in modo che **LOAD** viene seguito automaticamente da

GO TO numero di linea

SAVE nome **LINE** è equivalente a **SAVE** nome **LINE** 1, cosicchè, quando il programma è caricato, inizia a girare dal principio.

2. Bytes:

SAVE nome **CODE** byte d'inizio, numero dei bytes

memorizza *numero dei bytes* a partire da un certo *byte d'inizio*.

SAVE nome **SCREEN$**

è equivalente a

SAVE nome **CODE 16384,6912**

e memorizza l'immagine televisiva.

3. Matrici:

SAVE nome **DATA** lettera ()
oppure
SAVE nome **DATA** lettera$ ()

memorizza la matrice di nome **lettera** o **lettera$** (che ovviamente non ha alcuna relazione con *nome*).

VERIFY
Verifica che le informazioni memorizzate su nastro siano uguali alle informazioni in memoria. Se il confronto rivela delle differenze, si ha l'errore **R Tape loading error**.

1. Programmi e variabili:

VERIFY nome

2. Bytes:

VERIFY nome **CODE** byte d'inizio, numero dei bytes

Se i bytes **nome** su nastro sono in numero maggiore di **numero dei bytes**, si ha un errore di tipo **R**, altrimenti esegue la verifica a partire dal **byte d'inizio.**

VERIFY nome **CODE** byte d'inizio
confronta i bytes **nome** su nastro con quelli in memoria a partire dal **byte d'inizio.**

VERIFY nome **CODE**

confronta i bytes **nome** sul nastro con quelli in memoria che hanno gli stessi indirizzi che occupavano prima di essere memorizzati.

VERIFY nome **SCREEN$**

è equivalente a

VERIFY nome **CODE 16384,6912**

ma, in condizioni normali, non funzionerà, e darà quindi errore.

3. Matrici:

VERIFY nome **DATA** lettera ()
oppure
VERIFY nome **DATA** lettera $ ()

confronta la matrice **nome** su nastro con la matrice **lettera** o **lettera$** in memoria.

LOAD
Carica delle informazioni da nastro, cancellando le vecchie informazioni dalla memoria.

1. Programmi e variabili:

LOAD nome

cancella il vecchio programma e le vecchie variabili, e carica il nuovo programma

con le sue variabili, con nome *nome* dalla cassetta; se il programma era stato memorizzato con **SAVE** nome **LINE**, esegue un salto automatico. Se non c'è abbastanza spazio in memoria, il vecchio programma e le vecchie variabili non vengono cancellate, e si ha l'errore **4 Out of memory**

2. Bytes:

LOAD nome **CODE** byte d'inizio, numero dei bytes

Se i bytes *nome* sono di più che il *numero dei bytes*, ritorna un errore **R**, altrimenti li carica in memoria a cominciare dal **byte d'inizio**, cancellando il precedente contenuto di quella zona di memoria.

LOAD nome **CODE** byte d'inizio

carica i bytes *nome* dalla cassetta in memoria, a cominciare dall'indirizzo del *byte d'inizio*, e cancellando qualunque cosa occupasse prima quella zona di memoria.

LOAD nome **CODE**

carica i bytes *nome* dal nastro nella memoria agli stessi indirizzi che occupavano prima di essere memorizzati, e cancella i bytes in quella zona di memoria.

3. Matrici:

LOAD nome **DATA** lettera ()
oppure
LOAD nome **DATA** lettera $ ()

cancella qualunque eventuale matrice chiamata *lettera* o *lettera*$, e ne crea una nuova dello stesso nome, contenente i dati registrati su cassetta. Se non vi è abbastanza spazio per la nuova matrice, la vecchia matrice non viene cancellata e si ha errore **4 Out of memory.**

MERGE
Carica nuove informazioni dalla cassetta senza cancellare le vecchie conservate in memoria.

1. Programmi e variabili:

MERGE nome

fonde il programma **nome** con quello già in memoria, scrivendo sopra a tutte le variabili o linee comuni ai due programmi. Se non vi è abbastanza spazio in

memoria per tutto il vecchio programma e variabili, e per tutto il nuovo programma e relative variabili, si ha un errore **4 Out of memory**.

2. Bytes:

non utilizzabile

3. Matrici:

non utilizzabile

Esercizi

1. Fate una cassetta il cui primo programma, appena caricato, stampi un *menu*, cioè una lista degli altri programmi della cassetta, e vi chieda quale desiderate far girare, e quindi lo carichi e lo faccia girare.

2. Prendete il grafico dei pezzi degli scacchi dal capitolo 23, e scrivete **NEW**. Verificate che i caratteri grafici siano ancora definiti: comunque si perdono quando il calcolatore è spentoo; se li volete memorizzare, dovete salvarli come un'informazione di tipo byte, usando **SAVE** e **CODE**. Il modo più facile è di salvare tutti i 21 caratteri grafici definiti dall'utente con

SAVE "scacchi" CODE USR "a",21*8

seguito da

VERIFY "scacchi" CODE

Questo sistema è perfettamente analogo a quello usato per memorizzare l'immagine video. L'indirizzo del primo byte da salvare **USR "a"**, cioè l'indirizzo del primo degli 8 byte che determinano la maschera del carattere grafico definito dall'utente, e il numero di byte da salvare è ovviamente **21*8**, cioè 8 bytes per ognuno dei 21 caratteri. Per ricaricarli normalmente usereste

LOAD "scacchi" CODE

Comunque, se li dovete ricaricare in uno Spectrum con una differente quantità di memoria, o se avete spostato l'area dei caratteri grafici definiti dall'utente ad un indirizzo differente (cosa che potrete fare deliberatamente quando programmerete in modo più avanzato), dovrete essere più attenti e usare

LOAD "scacchi" CODE USR "a"

USR consente che i grafici debbano essere ricaricati in un indirizzo differente.

CAPITOLO 30

La stampante ZX printer

Sommario
LPRINT, LLIST, COPY

Nota: nessuno di questi comandi è standard in BASIC, per quanto **LPRINT** sia usato da qualche altro calcolatore, e così pure **LLIST**.

Se siete in possesso di una stampante ZX printer, avrete anche il suo manuale di istruzioni, che vi indica come collegarla. Questo capitolo vi spiega quali comandi BASIC si possono usare per farla funzionare.

I primi due, **LPRINT** e **LLIST**, sono esattamente identici a **LIST** e **PRINT**, solo che usano la stampante e non il video. Per aiutare la memoria, vi può interessare sapere che la "L", come molte altre cose, è un residuo storico. Quando il BASIC fu inventato, veniva usato generalmente con una telescrivente invece di una televisione, e quindi **PRINT** significava effettivamente "stampa" ("print", in inglese, significa appunto "stampa"). Essendo le telescriventi troppo lente per stampare grandi quantità di dati, i calcolatori venivano collegati anche a stampanti veloci, che stampavano tutta una linea alla volta, chiamate anche "line-printer", donde "L" davanti ai comandi di "print". Provate questo programma esempio:

```
10 LPRINT "Questo programma" . '
20 LLIST
30 LPRINT ' "stampa il set dei caratteri." '
40 FOR n=32 TO 255
50 LPRINT CHR$ n;
60 NEXT n
```

Il terzo comando, **COPY**, stampa una copia dello schermo televisivo. Per esempio, scrivete **LIST** per avere un listato sullo schermo, e poi scrivete

COPY

Notate che **COPY** non lavora con i listati che il calcolatore stampa automaticamente ogni volta che si preme **ENTER**, dato che vengono cancellati durante l'esecuzione di un comando. Per ottenere un listato su carta dovrete, o usare prima proprio il comando **LIST**, seguito da **COPY**, o usare direttamente **LLIST**.

Potete sempre fermare la stampante durante una stampa usando **BREAK** (**CAPS SHIFT** e **SPACE**).

Se eseguite i comandi per la stampante, senza che essa sia collegata, perdete tutti i dati in output ed il programma prosegue con la linea seguente. Provate questo:

```
10 FOR n=31 TO 0 step —1
20 PRINT AT 31—n,n; CHR$ (CODE "0"+n);
30 NEXT n
```

Quando il calcolatore vi chiede se desiderate lo scrolling, vedrete una fila diagonale di caratteri dall'angolo alto a destra fino alla base dello schermo.

Cambiate **AT 31—n,n** alla linea 20 con **TAB n** verificate quindi che il programma funzioni come prima. Ora cambiate **PRINT** alla linea 20 con **LPRINT** questa volta non vi saranno **scroll?**, dato che non servono con la stampante. La fila di caratteri continuerà con le lettere dalla F alla O. Cambiate **TAB n** in **AT 31—n,n**, continuando ad usare **LPRINT** questa volta otterrete una sola linea di simboli. La differenza si spiega tenendo conto che i dati posti in uscita con la **LPRINT** non vengono stampati immediatamente, ma vengono prima sistemati in un buffer lungo una linea di stampa, che viene inviata dal calcolatore alla stampante soltanto quando si verifica una delle seguenti condizioni:
(i) il buffer è pieno,
(ii) dopo un comando **LPRINT** che non finisca con una virgola o un punto e virgola,
(iii) quando una virgola, un apostrofo ed un **TAB** richiedono una nuova linea, (iv) al termine di un programma, se vi è ancora qualcosa da stampare nel buffer.

Il punto (iii) spiega perchè **TAB** funziona come avete visto. Come per **AT**, la posizione di **LPRINT** (come la posizione di **PRINT**, ma per la stampante), viene trasformata nel relativo numero di colonna, ma mentre **TAB**, per le regole già viste richiede una nuova linea, **AT** non può mai richiederne una, dato che viene ignorato il numero di linea a cui verrebbe richiesta la stampa. **AT** non farà mai sì che una linea venga mandata alla stampante, ma permette di scrivere a piacere nel buffer di stampa.

Esercizio

1. Fate il grafico della funzione **SIN**, facendo girare il programma del capitolo 28, e quindi copiatelo su stampante.

CAPITOLO 31

Altre periferiche

Allo ZX Spectrum possono essere collegate anche altre periferiche.

Lo ZX Microdrive è un'unità di memoria di massa ad alta velocità, ed è molto più flessibile di un registratore a cassette. Lavora non solo con **SAVE, VERIFY, LOAD** e **MERGE**, ma anche con **PRINT, LIST, INPUT** e **INKEY$**.

Può essere usata una rete per collegare diversi Spectrum in modo che possano comunicare tra loro. Usando una rete, è anche possibile collegare vari Spectrum tra di loro in modo da usare un solo Microdrive. L'interfaccia RS232 è standard, e vi permette di collegare uno Spectrum a tastiere, stampanti, altri calcolatori e varie altre macchine, anche se esse non sono predisposte per il collegamento ad esso. L'uso delle periferiche menzionate richiederà altre parole chiave già presenti sulla tastiera, ma che non avrete modo di usare con lo Spectrum nella configurazione base: sono **OPEN #, CLOSE #, MOVE, ERASE, CAT** e **FORMAT**.

CAPITOLO 32

Uso delle porte di INPUT-OUTPUT

Sommario
OUT
IN

Il processore può leggere e scrivere nella memoria RAM usando **PEEK** e **POKE**. In verità non si può scrivere in una memoria ROM, nè leggere in una memoria inesistente, ma il microprocessore non lo sa; esso sa solo che ci sono 65536 indirizzi di memoria, e che si può leggere un byte da ognuno, anche se non c'è alcun byte da leggere, e che si può scrivere un byte in ognuno, anche se va perso. In modo completamente analogo ci sono 65536 di quelle che sono chiamate *porte di input e output*, o *porte di I/O;* esse sono usate dal processore per comunicare con il mondo esterno, ovvero con cose tipo la tastiera, la stampante, ecc., e possono essere usate in BASIC tramite le funzioni **IN** e **OUT**.

IN è la funzione corrispondente a **PEEK**.

IN indirizzo

ha un solo argomento, l'indirizzo della porta, e ritorna il byte letto da quella porta.

OUT è il comando corrispondente a **POKE**.

OUT indirizzo, valore

scrive il dato valore alla porta dell'indirizzo specificato. In che modo l'indirizzo venga interpretato dipende molto dalla configurazione del calcolatore; spesso indirizzi diversi indicheranno la stessa destinazione. Sullo Spectrum è comodo immaginare che l'indirizzo sia scritto in codice binario, dato che i singoli bit tendono, per così dire, a lavorare indipendentemente. Vi sono 16 bits, che si suole chiamare (A sta per indirizzo):

A15, A14, A13, A12,......, A2, A1, A0

Notate che A0 è il primo bit, A1 il secondo, A2 il terzo e così via. I bit A0, A1, A2, A3, A4 sono significativi per lo Spectrum. Nelle operazioni di I/O sono normalmente a 1. Se qualcuno di essi è a 0, indica la richiesta di una particolare funzione di I/O. Siccome il calcolatore non può fare più di una cosa alla volta, uno solo di questi bits può essere a 0. I bits A5, A6, A7 sono ignorati, quindi, se siete un mago dell'elettronica, potete usarli voi stessi. Per le vostre applicazioni particolari farete meglio ad usare gli indirizzi che valgono 1 meno di un multiplo di 32, in modo tale che A0,..., A4 siano tutti 1. I bits A8, A9,..., A15 sono usati per dare delle informazioni extra.

Il byte letto o scritto è formato da 8 bits, a cui si fa spesso riferimento (usando D per *data* cioè *dati*, informazioni) come D7, D6,..., D1, D0. Segue una lista degli indirizzi delle porte usate.

Esiste un insieme di indirizzi di input, che legge la tastiera e la presa EAR. La tastiera è divisa in otto mezzefile di cinque tasti ciascuna.

IN 65278 legge la mezzafila **CAPS SHIFT-V**
IN 65022 legge la mezzafila **A-G**
IN 64510 legge la mezzafila **Q-T**
IN 63486 legge la mezzafila **1-5**
IN 61438 legge la mezzafila **0-6**
IN 57342 legge la mezzafila **P-7**
IN 49150 legge la mezzafila **ENTER-H**
IN 32766 legge la mezzafila **SPACE-B**

(Questi indirizzi sono 254+256*(255-2↑n) con *n* che varia da 0 a 7).

Quando viene letto un byte, i bits da D0 a D4 sono associati ai cinque tasti della mezzafila specificata: D0 per il tasto più esterno, D4 per il tasto più vicino al centro. Il bit è 0 se il tasto è premuto, 1 se non è premuto. D6 è il valore alla presa EAR.

La porta d'indirizzo 254 in uscita controlla l'altoparlante (D4), la presa MIC (D3), e determina il colore del bordo (D2, D1 e D0).

La porta d'indirizzo 251 fa funzionare la stampante, sia in lettura che in scrittura: in lettura controlla se la stampante è pronta a stampare una nuova linea, e in scrittura manda la linea che deve essere stampata.

Le porte degli indirizzi 254, 247 e 239 sono usate per le periferiche addizionali nominate nel capitolo 31.

Fate girare questo programma:

```
10 FOR n=0 TO 7: REM numero della mezzafila
20 LET a=254+256*(255-2↑n)
30 PRINT AT 0,0; IN a: GO TO 30
```

e divertitevi a premere diversi tasti. Quando ne avrete premuti abbastanza in una mezzafila da annoiarvi, premete **BREAK** e scrivete

NEXT n

CAPITOLO 33

La memoria

Sommario
CLEAR

Nel profondo del calcolatore, ogni cosa è memorizzata come bytes, cioè come numeri da 0 a 255. Anche se voi pensate di aver memorizzato il prezzo della lana, o gli indirizzi dei vostri fornitori di concime, tutto è stato convertito in un insieme numerico di bytes, che solo il calcolatore riesce a vedere.

Ogni byte della memoria è associato ad un indirizzo, che è un numero da 0 a FFFFh (h significa *esadecimale*, e quindi un indirizzo può essere memorizzato in 2 bytes). Potete pensare alla memoria come a una lunga fila di scatole numerate, ognuna delle quali può contenere un byte. Non tutte le scatole sono dello stesso tipo in ogni modo; nella versione standard dello Spectrum, con 16K di RAM, le scatole da 8000h a FFFFh non possono contenere niente, le scatole da 4000h a 7FFFh sono scatole di tipo RAM, che potete ispezionare per conoscere ed alterare il contenuto, e quelle tra 0 e 3FFFh sono scatole ROM, che sono protette da un coperchio trasparente che non può essere aperto; dovete limitarvi a guardare quello che vi è stato scritto quando il calcolatore è stato costruito.

ROM	RAM	Non usata	
0	4000h =16384	8000h =32768	FFFFh =65535

Per ispezionare il contenuto di una scatola si usa la funzione **PEEK**, che ha come argomento l'indirizzo della scatola e ne ritorna il contenuto. Per esempio, il seguente programma stampa il contenuto dei primi 21 bytes della ROM, ed i relativi indirizzi:

```
10 PRINT "Indirizzo"; TAB 10; "Byte"
20 FOR a=0 TO 20
30 PRINT a TAB 10; PEEK a
40 NEXT a
```

Quanto vedrete non ha molto senso per voi: sono istruzioni per il processore, gli dicono cosa fare, analogamente al libro di cucina che la massaia usa per cucinare.

Per cambiare il contenuto di una scatola (solo se RAM), potete usare il comando **POKE**, nella forma:

POKE indirizzo, nuovo contenuto

dove *indirizzo* e *nuovo contenuto* sono espressioni numeriche. Per esempio, se scrivete

POKE 31000,57

il byte di indirizzo 31000 viene forzato ad assumere il valore 57; scrivete

PRINT PEEK 31000

e verificate che contenga 57. Provate a scrivervi altri valori, e controllate che non si crei confusione. Il nuovo valore del **POKE** deve essere compreso tra —255 e + 255, e, se è negativo, vi viene automaticamente sommato 256.

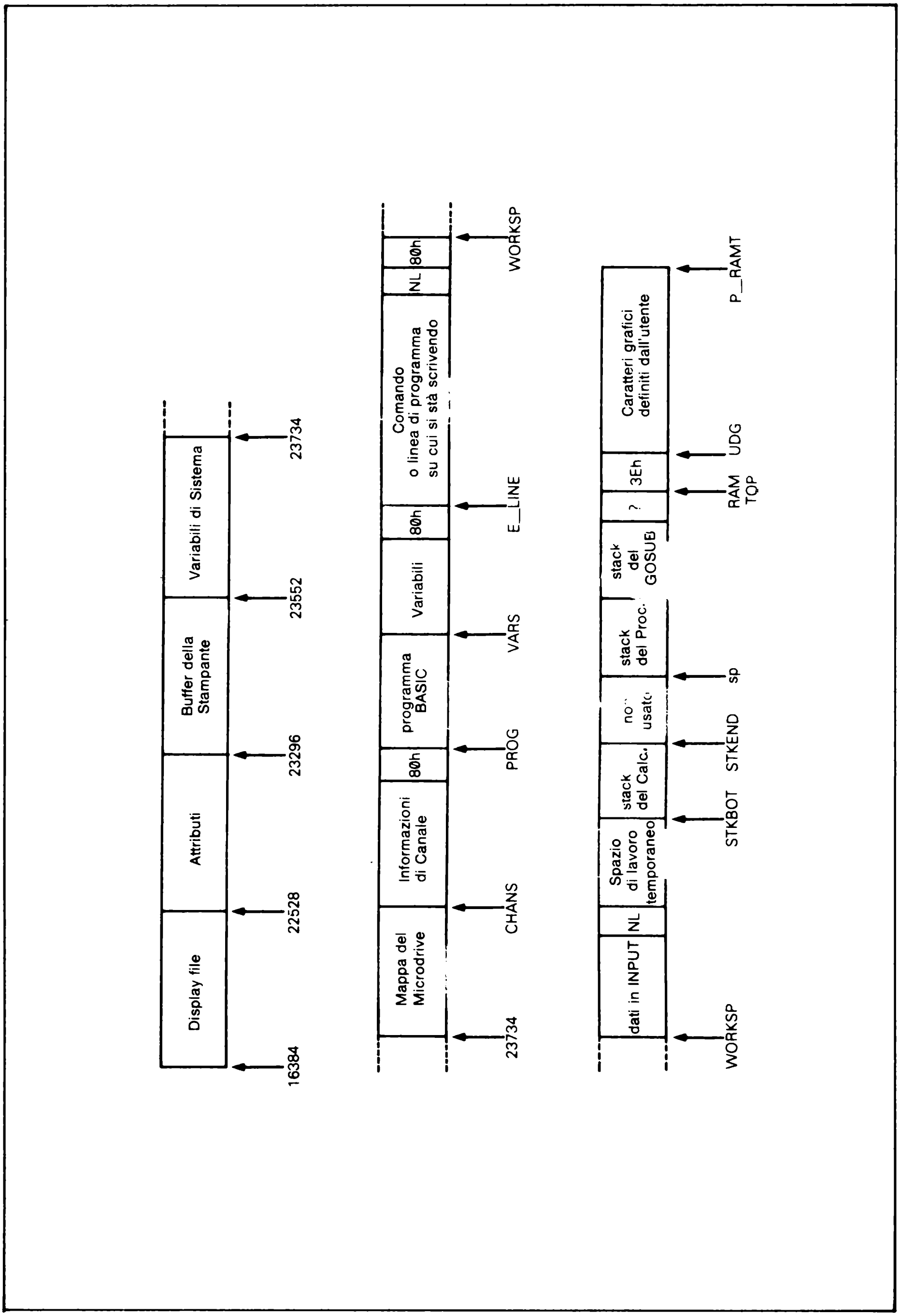
La possibilità di scrivere direttamente nelle locazioni di memoria del calcolatore vi dà, se siete in grado di usarla bene, un immenso potere di controllo, e se non lo siete, una grande possibilità di rovinare tutto. È molto facile, scrivendo un valore sbagliato in un indirizzo sbagliato, perdere grossi programmi che hanno richiesto ore di battitura. Fortunatamente è impossibile, anche in questo modo, danneggiare irreversibilmente il calcolatore. Nel resto di questo capitolo, prenderemo in considerazione in dettaglio l'uso della RAM: se non vi interessa, non preoccupatevi di procedere oltre.

La memoria è divisa in diverse aree, mostrate sullo schema, per memorizzare differenti tipi di informazioni. Le aree sono appena sufficienti per le informazioni che devono realmente contenere, e se ad un certo punto ne inserite delle altre, per esempio aggiungendo una variabile ad una linea di un programma, viene creato lo spazio necessario spostando tutto ciò che è sopra il limite dell'area da ampliare. Analogamente, se cancellate qualcosa, tutto viene spostato verso il basso.

La zona di memoria dove è conservata l'immagine video, chiamata *display file*, è organizzata in maniera piuttosto curiosa; difficilmente vi incaricherete di andarci a scrivere direttamente dentro. Ogni posizione di carattere sullo schermo è una matrice di 8*8 punti, ognuno dei quali può essere o 0 (carta), o 1 (inchiostro), e usando la notazione binaria, possiamo rappresentare la mascherina per l'inchiostro come 8 bytes, uno per ogni riga. Comunque questi 8 bytes non sono memorizzati consecutivamente; le righe corrispondenti nei 32 caratteri per ogni singola linea sono memorizzate insieme, come una serie di 32 bytes, secondo il percorso naturale del raggio di elettroni di scansione del video, da sinistra a destra. Dato che il quadro completo ha 24 linee composte ciascuna da 8 scansioni, potreste immaginarvi che l'insieme delle 172 scansioni sia memorizzato in ordine, una dopo l'altra, e vi sbagliereste. In realtà, all'inizio è memorizzata la prima linea di scansione di ognuna delle prime 7 righe, quindi la seconda linea, e così via fino all'ultima linea di scansione delle righe da 0 a 7; lo stesso accade per le righe da 8 a 15 e per quelle da 16 a 23. Si riesce a vedere questo modo di memorizzare quando si carica un'immagine video da cassetta. Provate riferendovi al capitolo 20.

Morale: se siete abituati con calcolatori che usano **PEEK** e **POKE** per scrivere direttamente sullo schermo, è meglio che vi abituiate a usare al loro posto **SCREEN$** e **PRINT AT**, e **PLOT** e **POINT** per la grafica.

Gli attributi, cioè i colori, ecc., di ogni posizione di carattere sono memorizzati nell'ordine sequenziale secondo il formato di **ATTR**.

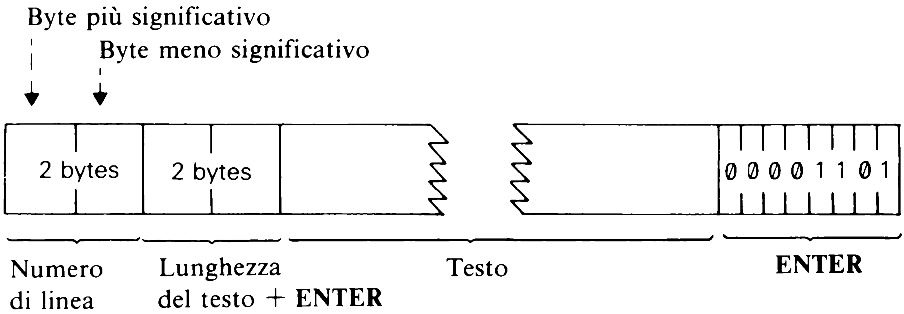

Display file
Attributi
Buffer della Stampante
Variabili di Sistema
16384
22528
23296
23552
23734
Mappa del Microdrive
Informazioni di Canale
8Øh
programma BASIC
Variabili
8Øh
Comando o linea di programma su cui si stà scrivendo
NL
8Øh
23734
CHANS
PROG
VARS
E_LINE
WORKSP
dati in INPUT
NL
Spazio di lavoro temporaneo
stack del Calc.
no
usat
stack del Proc.
stack del GOSUB
?
3Eh
Caratteri grafici definiti dall'utente
WORKSP
STKBOT
STKEND
sp
RAM TOP
UDG
P_RAMT

Il buffer della stampante conserva i caratteri destinati alla stampante stessa. Le variabili di sistema, tutte spiegate nel prossimo capitolo, contengono delle informazioni necessarie alla gestione interna. Per ora notate che ve ne sono alcune (CHANS, PROG, VARS, E_LINE e altre ancora), che indicano l'indirizzo limite delle varie aree di memoria. Le variabili di controllo non sono variabili BASIC, quindi i loro nomi non sono riconosciuti dal calcolatore, e servono soltanto a facilitarne l'identificazione.

La *mappa del Microdrive* è usata solo col Microdrive normalmente non contiene nulla.

Le informazioni di canale riguardano i dispositivi di input e output, cioè la tastiera (con la parte bassa dello schermo) per l'input, la parte alta dello schermo e la stampante per l'output.

Ogni linea di BASIC è nella forma:

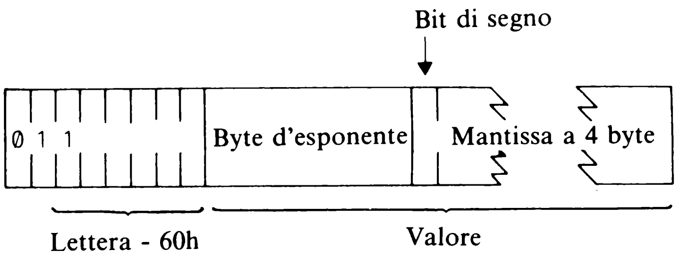


Notate che il numero di linea viene memorizzato nell'ordine in cui viene scritto, cioè col byte più significativo per primo, diversamente da tutti gli altri casi di numeri a due byte nello Z80 (il microprocessore dello ZX Spectrum).

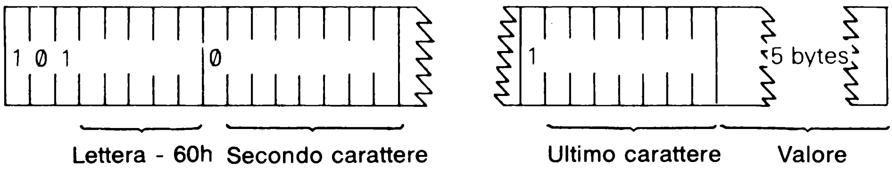
Una costante numerica in un programma, è seguita dalla sua scrittura binaria, usando il carattere **CHR$ 14** seguito da 5 bytes per il numero stesso.

Le variabili sono memorizzate con un formato diverso a seconda del tipo. Pensate al nome come se iniziasse con una lettera minuscola.

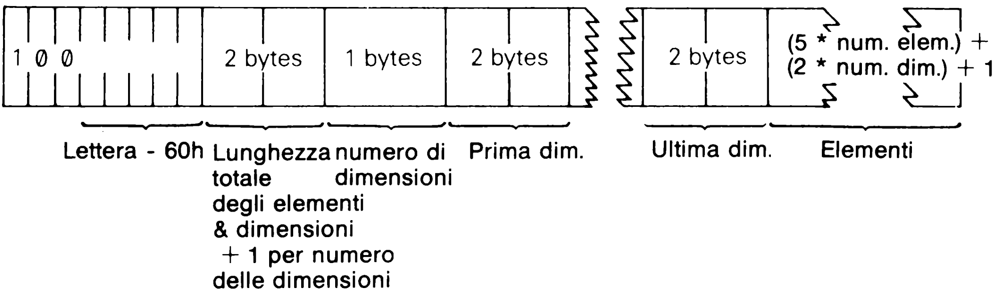
Variabili numeriche con il nome formato da una sola lettera:



Variabili numeriche con nome più lungo di una lettera:



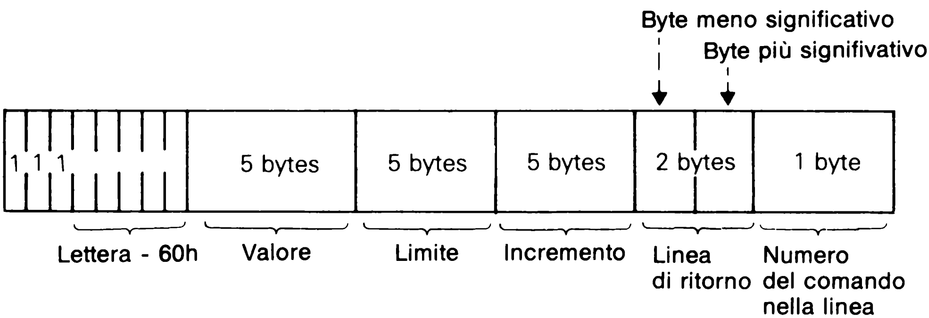
Matrici di numeri:



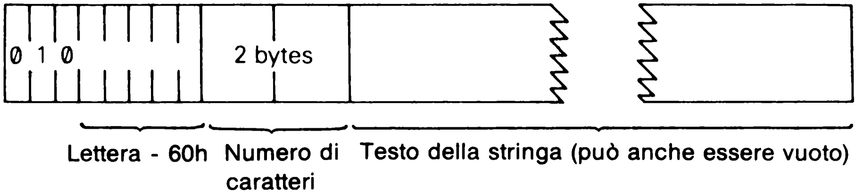
L'ordine degli elementi è:
primo, l'elemento di primo indice 1
poi, l'elemento di primo indice 2
poi, l'elememto di primo indice 3
e così via, per tutti i possibili valori del primo indice.

Gli elementi con lo stesso primo indice sono ordinati allo stesso modo secondo un secondo indice, e così via fino all'ultimo. Per esempio, gli elementi della matrice b(3,6) del capitolo 22, sono memorizzati nell'ordine b(1,1), b(1,2), b(1,3), b(1,4), b(1,5), b(1,6), b(2,1), b(2,2), ... b(2,6), b(3,1), b(3,2), ... b(3,6).

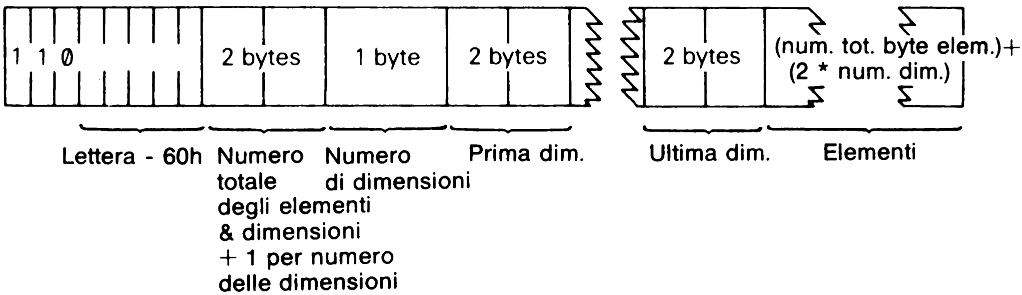
Variabile di controllo per un ciclo **FOR-NEXT**:



Stringa:



Matrici di caratteri (stringhe):



Lo stack del calcolatore contiene la maggior parte dei numeri usati da quella parte del BASIC che si occupa di eseguire le operazioni aritmetiche.

Non usato (SPARE) riguarda i segmenti di memoria non utilizzati.

Lo stack del processore è quello proprio dello Z80, usato per gli indirizzi di ritorno, ecc. Lo stack della **GOSUB** è stato spiegato nel capitolo 5.

Il byte puntato da RAMTOP è il limite dell'area di lavoro BASIC. **NEW** pulisce la RAM solo fino a questo indirizzo, e non cancella quindi i caratteri grafici definiti dall'utente. RAMTOP può essere cambiato usando l'indirizzo desiderato come argomento numerico in un comando **CLEAR**:

CLEAR nuovo RAMTOP

che
(i) cancella tutte le variabili
(ii) cancella il display file (come **CLS**)
(iii) resetta la posizione di **PLOT** all'angolo in basso a sinistra
(iv) esegue **RESTORE**
(v) pulisce lo stack della **GOSUB** e lo sposta al nuovo RAMTOP, che può essere soltanto tra lo stack del calcolatore e la fine fisica della RAM, altrimenti il comando **CLEAR** non lo sposta.

RUN esegue sempre **CLEAR** senza spostare RAMTOP.

Quest'uso di **CLEAR** permette di spostare RAMTOP in su, per fare spazio ad un programma BASIC piuttosto lungo, prendendo anche la zona dei caratteri grafici definiti dall'utente, o in giù, per riservare una parte di RAM che non venga modificata dal **NEW**, utile ad esempio per memorizzare programmi in linguaggio macchina.

Per rendervi conto di cosa accade quando la memoria BASIC si riempie, eseguite **CLEAR 23800** dopo **NEW**. Cominciate a scrivere un programma e noterete che quasi subito il calcolatore non accetterà più linee ed emetterà un suono per indicare che non c'è più spazio per il programma BASIC. Sono possibili due errori, che significano più o meno la stessa cosa: **4 Memory full** e **G No room for line**.

Se scrivete una linea di programma lunga più di 23 linee, non sarete più in grado di vedere cosa scrivete, anche se verranno accettati altri caratteri e sarà emesso un suono per scoraggiarvi dal continuare quella linea. Oltretutto non vi è alcun vantaggio pratico a scrivere linee di programma eccessivamente lunghe.

La lunghezza del suono può essere regolata scrivendo la durata desiderata nella locazione 23608. La lunghezza normale è 64.

Ogni numero, tranne 0, può essere scritto in modo inequivocabile come

$$\pm m*2^e$$

dove ± è il segno

m è la *mantissa*, compresa tra 1/2 e 1 (1 escluso)

e è l'esponente, un numero intero, positivo o negativo.

Supponete di scrivere m come numero binario. Dato che è una frazione, avrà la *virgola binaria*, così come un numero decimale ha la virgola decimale, e sarà quindi una frazione binaria (anche un numero decimale può essere convertito in frazione), così in binario si scrive

.1 per un mezzo

.01 per un quarto

.11 per tre quarti

.000110011001100110011... per un decimo e così via. Il numero m, essendo minore di 1, non ha bits prima del punto decimale, e dato che è almeno 1/2, il primo bit dopo il punto è sicuramente 1.

Il calcolatore memorizza i numeri in 5 bytes, col seguente procedimento

(i) prima scrive i primi 8 bits della mantissa nel secondo byte, e sappiamo che il primo bit sarà sempre 1, i secondi 8 bits nel terzo byte, i terzi nel quarto byte, i quarti nel quinto

(ii) sostituisce il primo bit nel secondo byte (sappiamo che è 1) col bit di segno, 0 per il +, 1 per il —

(iii) scrive l'esponente +128 nel primo byte. Per esempio, se il numero è 1/10:

$$1/10=4/5*2^{-3}$$

Quindi la mantissa m vale .11001100110011001100110011001100 in binario, e

dato che il trentatreesimo bit è 1, il trentaduesimo sarà arrotondato per eccesso da 0 a 1; e vale —3.

Applicando le regole date ai cinque bytes

questo 0 indica che il numero è positivo

0111 1101	0100 1100	1100 1100	1100 1100	1100 1101

−3+128   mantissa di 4/5 salvo per il primo bit, che sappiamo vale 1

Esiste un altro modo per memorizzare un numero intero compreso tra —65535 e +65535:

(i) il primo byte è 0

(ii) il secondo byte è 0 per un numero positivo, FFh per un numero negativo

(iii) il terzo ed il quarto byte sono quelli meno e più significativi del numero (o il numero +131072 se è negativo)

(iv) il quinto byte è 0.

CAPITOLO 34

Le variabili di sistema

Le variabili di sistema sono variabili non riconosciute dal BASIC (il nome che daremo loro in questo capitolo è puramente mnemonico per facilitarne l'identificazione) che contengono dei dati per la gestione interna del sistema. Sono contenute nelle aree di memoria dall'indirizzo 23552 all'indirizzo 23733, e possono essere lette ed alterate utilmente con la **PEEK** e la **POKE**.
Segue una tabella di tutte le variabili di sistema, con il loro nome, indirizzo, la spiegazione del significato, e una nota, in prima colonna, che indica se:

X Le variabili non devono assolutamente essere alterate, perchè si potrebbe causare l'arresto del sistema.
N Alterare la variabile non ha un effetto distruttivo sul sistema
Il numero che segue la nota è il numero di bytes della variabile. Per la variabili a 2 bytes il primo è il meno significativo, diversamente da quanto potreste attendervi. Quindi, per alterare il valore della variabile a 2 bytes ad indirizzo **n** in **v**, usate:

POKE n,v—256*INT (v/256)
POKE n+1,INT (v/256)

e per ottenere il suo valore usate l'espressione:

PEEK n+256*PEEK (n+1)

Note	*Indirizzi*	*Nome*	*Contenuto*
N8	23552	KSTATE	Usata nella lettura della tastiera.
N1	23560	LAST K	Ultimo tasto premuto.
1	23561	REPDEL	Tempo, in cinquantesimi di secondo, per cui è necessario tenere premuto un tasto prima che inizi a ripetersi. All'inizio è 35, ma può essere cambiato a piacere.
1	23562	REPPER	Ritardo tra una ripetizione e la successiva di un tasto tenuto premuto; inizialmente 5.
N2	23563	DEFADD	Normalmente 0; quando una funzione definita dall'utente viene calcolata contiene l'indirizzo degli argomenti.
N1	23565	K DATA	Contiene il secondo byte dei controlli di colore introdotti da tastiera.
N2	23566	TVDATA	Memorizza i bytes dei controlli di colore, **AT** e **TAB** per il controllo della televisione.

Note	*Indirizzi*	*Nome*	*Contenuto*
X38	23568	STRMS	Indirizzi dei canali associati ai flussi di dati.
2	23606	CHARS	Indirizzo del set dei caratteri meno 256. Il set inizia con spazio e termina con il simbolo di copyright, risiede normalmente in ROM ma può essere ridefinito in RAM. Viene usato il set puntato da CHARS.
1	23608	RASP	Lunghezza del campanello di allarme.
1	23609	PIP	Lunghezza del clic dei tasti.
1	23610	ERR NR	Numero di messaggio meno 1. All'inizio è 255 (per 0 — 1) quindi **PEEK 23610** ritorna 255.
X1	23611	FLAGS	Flag vari di controllo del BASIC.
X1	23612	TV FLAG	Flag associati col video
X2	23613	ERR SP	Indirizzo dell'elemento dello stack del processore da usare in caso di errore.
N2	23615	LIST SP	Indirizzo di ritorno dopo un listato automatico.
N1	23617	MODE	Indica se il cursore è nel modo K, L, C, E o G.
2	23618	NEWPPC	Linea a cui saltare.
1	23620	NSPPC	Numero di comando della linea a cui saltare. Alterare prima NEWPPC e poi NSPPC causa un salto ad un specificato comando in una linea.
2	23621	PPC	Numero di linea che contiene il comando in esecuzione.
1	23623	SUBPPC	Posizione del comando in esecuzione relativamente alla linea in cui si trova.
1	23624	BORDCR	Colore del bordo per 8; contiene anche gli attributi usati normalmente per la parte bassa dello schermo.
2	23625	E PPC	Numero della linea corrente (puntato dal cursore di programma)
X2	23627	VARS	Indirizzo delle variabili
N2	23629	DEST	Indirizzo della variabile in assegnazione.
X2	23631	CHANS	Indirizzo del canale dei dati.
X2	23633	CURCHL	Indirizzo delle informazioni usate al momento per input e output.
X2	23635	PROG	Indirizzo del programma BASIC.
X2	23637	NXTLIN	Indirizzo della prossima linea del programma.
X2	23639	DATADD	Indirizzo del terminatore dell'ultimo elemento della DATA.
X2	23641	E LINE	Indirizzo del comando in introduzione.
2	23643	K CUR	Indirizzo del cursore.

Note	Indirizzi	Nome	Contenuto
X2	23645	CH ADD	Indirizzo del prossimo carattere da interpretare: il carattere dopo l'argomento di **PEEK** o il **NEW-LINE** al termine di un comando **POKE**.
2	23647	X PTR	Indirizzo del carattere dopo l'indicatore di errore **?** .
X2	23649	WORKSP	Indirizzo dell'area di lavoro momentanea.
X2	23651	STKBOT	Indirizzo base dello stack del calcolatore.
X2	23653	STKEND	Indirizzo di partenza dello spazio non utilizzato.
N1	23655	BREG	Registro b del calcolatore.
N2	23656	MEM	Indirizzo della zona usata come memoria nel calcolatore (di solito MEMBOT, ma non sempre).
1	23658	FLAGS2	Altri flags.
X1	23659	DF SZ	Numero di linee (compreso quella bianca) nella parte bassa dello schermo.
2	23660	S TOP	Numero della prima linea di programma nel listing automatico.
2	23662	OLDPPC	Numero di linea a cui deve saltare **CONTINUE**.
1	23664	OSPCC	Numero del comando della linea a cui deve saltare **CONTINUE**.
N1	23665	FLAGX	Flags vari.
N2	23666	STRLEN	Lunghezza del tipo della stringa di destinazione in assegnazione.
N2	23668	T ADDR	Indirizzo dell'elemento successivo nella tavola di sintassi (di uso molto raro).
2	23670	SEED	Seme per **RND**, alterato da **RANDOMIZE**.
3	23672	FRAMES	3 byte del contatore di quadri video, con il meno significativo per primo, incrementato ogni 20 millisecondi. Vedere capitolo 27.
2	23675	UDG	Indirizzo del primo carattere grafico definito dall'utente. Può essere cambiato, per esempio, per avere più spazio e chiaramente meno caratteri grafici.
1	23677	COORDS	Coordinata x dell'ultimo punto disegnato.
1	23678		Coordinata y dell'ultimo punto disegnato.
1	23679	P POSN	33 - numero di colonna della posizione di stampa della stampante.
1	23680	PR CC	Byte meno significativo dell'indirizzo della posizione di stampa nel buffer della stampante (posizione a cui dovrà stampare **LPRINT**).
1	23681		Non usata.

Note	*Indirizzi*	*Nome*	*Contenuto*
2	23682	ECHO E	33 - numero di colonna, 24 - numero di linea della posizione finale del buffer di input nella parte bassa dello schermo.
2	23684	DF CC	Indirizzo della posizione di stampa nel display file.
2	23686	DFCCL	Analoga a DF CC per la parte bassa dello schermo.
X1	23688	S POSN	33- numero di colonna della posizione di stampa.
X1	23689		24 - numero di linea della posizione di stampa.
X2	23690	SPOSNL	Analogo a S POSN per la parte bassa dello schermo.
1	23692	SCR CT	Contatore di scroll; è sempre 1 più il numero di scrolls che devono essere eseguiti prima di fermare e chiedere **scroll?**. Se viene continuamente aggiornato con un numero più grosso di 1 (di solito 255), lo schermo continua a funzionare senza mai fermarsi per chiedere lo **scroll?**.
1	23693	ATTR P	Colori correnti permanenti, ecc., decisi dai comandi di colore.
1	23694	MASK P	Usata per la trasparenza: ogni bit che è 1 indica che il corrispondente attributo non deve essere preso da ATTR P, ma deve essere lasciato inalterato ciò che è presente sullo schermo.
N1	23695	ATTR T	Colori correnti e temporanei.
N1	23696	MASK T	Analoga a MASK P, ma per i colori temporanei.
1	23697	P FLAG	Altri flags.
N30	23698	MEMBOT	Area di memoria del calcolatore, usata per conservare numeri che non possono essere posti utilmente nello stack del calcolatore.
2	23728		Non usata.
2	23730	RAMTOP	Indirizzo dell'ultimo byte nell'area di sistema BASIC.
2	23732	P-RAMT	Indirizzo dell'ultimo byte fisico della RAM.

Per esempio, questo programma lista i primi 22 bytes dell'area variabili:

```
10 FOR n=0 TO 21
20 PRINT PEEK (PEEK 23627+256*PEEK 23628+n)
30 NEXT n
```

Cercate di capire a quale variabile appartengano i bytes, tenendo presenti i valori della variabile di controllo **n**.

Cambiate la linea 20 in:

20 PRINT PEEK (23755+n)

e avrete la stampa dei primi 22 bytes dell'area programmi: confrontate quello che ottenete col programma stesso.

CAPITOLO 35

Uso del linguaggio macchina

Sommario
USR con argomento numerico

Questo capitolo è stato scritto per coloro che capiscono il *linguaggio macchina dello Z80*, cioè il set di istruzioni usate dal processore. Se non lo conoscete, e vi piacerebbe conoscerlo, potete leggere uno dei numerosi libri scritti sull'argomento. Se non avete mai programmato in nessun linguaggio *assembler*, vi converrà acquistarne uno per principianti, non importa anche se non nomina lo Spectrum.

I programmi del tipo di cui si tratta in questo capitolo vengono normalmente scritti in un linguaggio di programmazione chiamato *assembler*, che, per quanto sia molto simile al linguaggio macchina, è comunque di uso non troppo difficoltoso dopo un pò di pratica, deve essere tradotto in codici binari, prima di poter essere eseguito. L'Appendice A riporta una lista delle istruzioni del linguaggio macchina dello Z80, contro i codici dei tasti. La traduzione dall'assembler in linguaggio macchina, viene normalmente eseguita da un programma chiamato assemblatore che può essere comprato su cassetta. Se comunque non avete una grande necessità di lavorare col linguaggio macchina, o vi limitate a scrivere programmi corti, potete eseguire comodamente l'assemblaggio manuale.

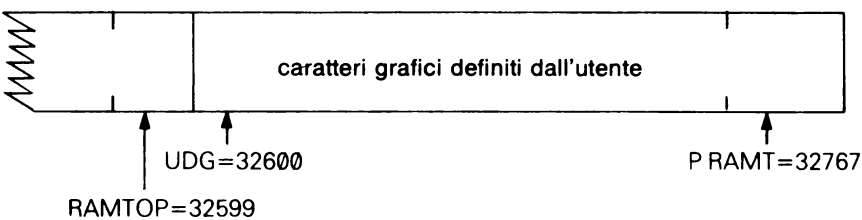
Per esempio, il programma:

```
ld bc, 99
ret
```

che carica 99 nella coppia di registri bc, si traduce in linguaggio macchina nei quattro bytes 1, 99, 0 (per ld bc, 99) e 201 (per ret). Se cercate nell'Appendice A 1 e 201, troverete ld bc, NN (dove NN rappresenta un qualunque numero a due bytes) e ret.

Il passo successivo è di introdurre il linguaggio macchina nel calcolatore, funzione anche questa generalmente svolta dall'assemblatore. In primo luogo è necessario stabilire in che locazione di memoria andrà sistemato, e sullo Spectrum la cosa migliore è di metterlo tra l'area del BASIC e l'area dei caratteri grafici definiti dall'utente.

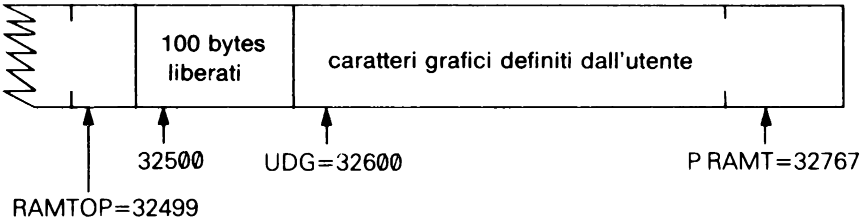
Supponete di avere uno Spectrum da 16K; all'accensione la cima della RAM è



ma potete spostare RAMTOP con

CLEAR 32499

e ottenere che i 100 bytes, a partire dall'indirizzo 32500, non siano più utilizzati dal sistema.



Per inserire i codici in memoria, potete usare un programma tipo il seguente:

```
10 LET a=32500
20 READ n: POKE a,n
30 LET a=a+1: GO TO 20
40 DATA 1,99,0,201
```

(Che termina col messaggio **E Out of DATA** quando ha terminato i 4 bytes specificati).

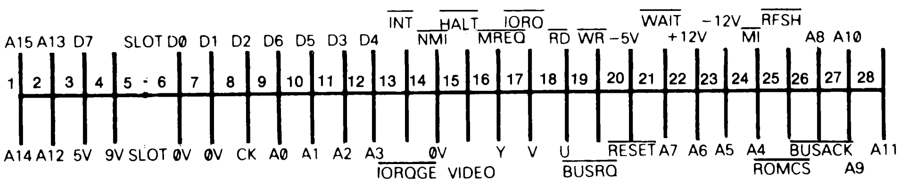
Una volta introdotto il programma in liguaggio macchina, può essere fatto girare con la funzione **USR**, che usa come argomento numerico l'indirizzo di partenza. Usandola così, la funzione **USR** ritorna il valore della coppia di registri bc, al termine del programma in linguaggio macchina. Quindi, se eseguite

PRINT USR 32500

ottenete la stampa di 99.

L'indirizzo di ritorno al BASIC è memorizzato al solito modo, e si ottiene quindi con l'istruzione ret dello Z80. In una routine in linguaggio macchina dovete stare attenti a non usare i registri iy ed i.

Lo Spectrum può essere interfacciato col mondo esterno quasi allo stesso modo di uno Z80, grazie ai bus dei dati, degli indirizzi e di controllo riportati al connettore sul retro. A volte, l'hardware dello Spectrum può essere di aiuto. Ecco uno schema delle connessioni del connettore:



Un programma in linguaggio macchina può essere memorizzato come un'informazione di tipo byte, specificando l'indirizzo di partenza e la lunghezza; consultate per questo il capitolo 6 sull'uso del registratore a cassette. Per memorizzare il programma esempio, si usa

SAVE "nome" **CODE 32500,4**

Non si può fare in modo che un programma in assembler si lanci automaticamente appena caricato, ma è possibile lanciarlo da un programma BASIC, che parte automaticamente. Quindi per avere il programma esempio eseguito appena caricato, è necessario passare per un programma BASIC del tipo:

10 LOAD "" CODE 32500,4
20 PRINT USR 32500

Scivetelo, ed eseguite

SAVE "nome" LINE

e poi

SAVE "xxxx" CODE 32500,4

riavvolgete e scrivete

LOAD "nome"

che caricherà e farà eseguire il programma BASIC, il quale provvederà a sua volta, a far caricare e a far eseguire il programma in linguaggio macchina.

APPENDICE A

Il set di caratteri

Tutto il set di caratteri dello ZX Spectrum è listato di seguito in ordine crescente di codice, riportando sia il codice decimale che quello esadecimale. In più sono state riportate le istruzioni che verrebbero eseguite dallo Z80, se interpretasse il codice di un carattere come un suo codice istruzione, e siccome molte istruzioni dello Z80 sono formate da due bytes, il primo dei quali è CBh oppure EDh, sono anche state riportate le istruzioni corrispondenti, supponendo che il codice del carattere sia preceduto da uno dei due prefissi.

Codice	*Carattere*	*Hex*	*Assembler Z80*	*- dopo CB*	*- dopo ED*
0	non usati	00	nop	rlc b	
1	non usati	01	ld bc,NN	rlc c	
2	non usati	02	ld (bc),a	rlc d	
3	non usati	03	inc bc	rlc e	
4	non usati	04	inc b	rlc h	
5	non usati	05	dec b	rlc l	
6	virgola della **PRINT**	06	ld b,N	rlc (hl)	
7	**EDIT**	07	rlca	rlc a	
8	cursore a sinistra	08	ex af,af'	rrc b	
9	cursore a destra	09	add hl,bc	rrc c	
10	cursore su	0A	ld a,(bc)	rrc d	
11	cursore giù	0B	dec bc	rrc e	
12	**DELETE**	0C	inc c	rrc h	
13	**ENTER**	0D	dec c	rrc l	
14	numero	0E	ld c,N	rrc (hl)	
15	non usato	0F	rrca	rrc a	
16	car. controllo **INK**	10	djnz DIS	rl b	
17	car. controllo **PAPER**	11	ld de,NN	rl c	
18	car. controllo **FLASH**	12	ld (de),a	rl d	
19	car. controllo **BRIGHT**	13	inc de	rl e	
20	car. controllo **INVERSE**	14	inc d	rl h	
21	car. controllo **OVER**	15	dec d	rl l	
22	carica **AT**	16	ld d,N	rl (hl)	

Codice	*Carattere*	*Hex*	*Assembler Z80*	*- dopo CB*	*- dopo ED*
23	car. controllo **TAB**	17	rla	rl a	
24		18	jr DIS	rr b	
25		19	add hl,de	rr c	
26		1A	ld a,(de)	rr d	
27		1B	dec de	rr e	
28	non usati (24–31)	1C	inc e	rr h	
29		1D	dec e	rr l	
30		1E	ld e,N	rr (hl)	
31		1F	rra	rr a	
32	spazio	20	jr nz,DIS	sla b	
33	!	21	ld hl,NN	sla c	
34	"	22	ld (NN),hl	sla d	
35	#	23	inc hl	sla e	
36	$	24	inc h	sla h	
37	%	25	dec h	sla l	
38	&	26	ld h,N	sla (hl)	
39	'	27	daa	sla a	
40	(	28	jr z,DIS	sra b	
41	)	29	add hl,hl	sra c	
42	*	2A	ld hl,(NN)	sra d	
43	+	2B	dec hl	sra e	
44	,	2C	inc l	sra h	
45	—	2D	dec l	sra l	
46	.	2E	ld l,N	sra (hl)	
47	/	2F	cpl	sra a	
48	0	30	jr nc,DIS		
49	1	31	ld sp,NN		
50	2	32	ld (NN),a		
51	3	33	inc sp		
52	4	34	inc (hl)		
53	5	35	dec (hl)		
54	6	36	ld (hl),N		
55	7	37	scf		
56	8	38	jr c, DIS	srl b	
57	9	39	add hl,sp	srl c	
58	:	3A	ld a,(NN)	srl d	
59	;	3B	dec sp	srl e	
60	<	3C	inc a	srl h	
61	=	3D	dec a	srl l	
62	>	3E	ld a,N	srl (hl)	
63	?	3F	ccf	srl a	
64	@	40	ld b,b	bit 0,b	in b,(c)

Codice	*Carattere*	*Hex*	*Assembler Z80*	*- dopo CB*	*- dopo ED*
65	A	41	ld b,c	bit 0,c	out (c),b
66	B	42	ld b,d	bit 0,d	sbc hl,bc
67	C	43	ld b,e	bit 0,e	ld (NN),bc
68	D	44	ld b,h	bit 0,h	neg
69	E	45	ld b,l	bit 0,l	retn
70	F	46	ld b,(hl)	bit 0,(hl)	im 0
71	G	47	ld b,a	bit 0,a	ld i,a
72	H	48	ld c,b	bit 1,b	in c,(c)
73	I	49	ld c,c	bit 1,c	out (c),c
74	J	4A	ld c,d	bit 1,d	adc hl,bc
75	K	4B	ld c,e	bit 1,e	ld bc,(NN)
76	L	4C	ld c,h	bit 1,h	
77	M	4D	ld c,l	bit 1,l	reti
78	N	4E	ld c,(hl)	bit 1,(hl)	
79	O	4F	ld c,a	bit 1,a	ld r,a
80	P	50	ld d,b	bit 2,b	in d,(c)
81	Q	51	ld d,c	bit 2,c	out (c),d
82	R	52	ld d,d	bit 2,d	sbc hl,de
83	S	53	ld d,e	bit 2,e	ld (NN),de
84	T	54	ld d,h	bit 2,h	
85	U	55	ld d,l	bit 2,l	
86	V	56	ld d,(hl)	bit 2,(hl)	im 1
87	W	57	ld d,a	bit 2,a	ld a,i
88	X	58	ld e,b	bit 3,b	in e,(c)
89	Y	59	ld e,c	bit 3,c	out (c),e
90	Z	5A	ld e,d	bit 3,d	adc hl,de
91	[	5B	ld e,e	bit 3,e	ld de,(NN)
92	/	5C	ld e,h	bit 3,h	
93	]	5D	ld e,l	bit 3,l	
94	↑	5E	ld e,(hl)	bit 3,(hl)	im 2
95	—	5F	ld e,a	bit 3,a	ld a,r
96	£	60	ld h,b	bit 4,b	in h,(c)
97	a	61	ld h,c	bit 4,c	out (c),h
98	b	62	ld h,d	bit 4,d	sbc hl,hl
99	c	63	ld h,e	bit 4,e	ld (NN),hl
100	d	64	ld h,h	bit 4,h	
101	e	65	ld h,l	bit 4,l	
102	f	66	ld h,(hl)	bit 4,(hl)	
103	g	67	ld h,a	bit 4,a	rrd
104	h	68	ld l,b	bit 5,b	in l,(c)
105	i	69	ld l,c	bit 5,c	out (c),l
106	j	6A	ld l,d	bit 5,d	adc hl,hl

Codice	*Carattere*		*Hex*	*Assembler Z80*	*- dopo CB*	*- dopo ED*
107	k		6B	ld l,e	bit 5,e	ld hl,(NN)
108	l		6C	ld l,h	bit 5,h	
109	m		6D	ld l,l	bit 5,l	
110	n		6E	ld l,(hl)	bit 5,(hl)	
111	o		6F	ld l,a	bit 5,a	rld
112	p		70	ld (hl),b	bit 6,b	in f,(c)
113	q		71	ld (hl),c	bit 6,c	
114	r		72	ld (hl),d	bit 6,d	sbc hl,sp
115	s		73	ld (hl),e	bit 6,e	ld (NN),sp
116	t		74	ld (hl),h	bit 6,h	
117	u		75	ld (hl),l	bit 6,l	
118	v		76	halt	bit 6,(hl)	
119	w		77	ld (hl),a	bit 6,a	
120	x		78	ld a,b	bit 7,b	in a,(c)
121	y		79	ld a,c	bit 7,c	out (c),a
122	z		7A	ld a,d	bit 7,d	adc hl, sp
123	{		7B	ld a,e	bit 7,e	ld sp,(NN)
124	|		7C	ld a,h	bit 7,h	
125	}		7D	ld a,l	bit 7,l	
126	~		7E	ld a,(hl)	bit 7,(hl)	
127	©		7F	ld a,a	bit 7,a	
128	□		80	add a,b	res 0,b	
129	▝		81	add a,c	res 0,c	
130	▘		82	add a,d	res 0,d	
131	▀		83	add a,e	res 0,e	
132	▗		84	add a,h	res 0,h	
133	▐		85	add a,l	res 0,l	
134	▚		86	add a,(hl)	res 0,(hl)	
135	▜		87	add a,a	res 0,a	
136	▖		88	adc a,b	res 1,b	
137	▞		89	adc a,c	res 1,c	
138	▌		8A	adc a,d	res 1,d	
139	▛		8B	adc a,e	res 1,e	
140	▄		8C	adc a,h	res 1,h	
141	▟		8D	adc a,l	res 1,l	
142	▙		8E	adc a,(hl)	res 1,(hl)	
143	█		8F	adc a,a	res 1,a	
144	(a)	caratteri grafici definiti dall'utente	90	sub b	res 2,b	
145	(b)		91	sub c	res 2,c	
146	(c)		92	sub d	res 2,d	
147	(d)		93	sub e	res 2,e	
148	(e)		94	sub h	res 2,h	

Codice	*Carattere*		*Hex*	*Assembler Z80*	*- dopo CB*	*- dopo ED*
149	(f)		95	sub l	res 2,l	
150	(g)		96	sub (hl)	res 2,(hl)	
151	(h)		97	sub a	res 2,a	
152	(i)		98	sbc a,b	res 3,b	
153	(j)		99	sbc a,c	res 3,c	
154	(k)		9A	sbc a,d	res 3,d	
155	(l)	caratteri	9B	sbc a,e	res 3,e	
156	(m)	grafici	9C	sbc a,h	res 3,h	
157	(n)	definiti	9D	sbc a,l	res 3,l	
158	(o)	dall'utente	9E	sbc a,(hl)	res 3,(hl)	
159	(p)		9F	sbc a,a	res 3,a	
160	(q)		A0	and b	res 4,b	ldi
161	(r)		A1	and c	res 4,c	cpi
162	(s)		A2	and d	res 4,d	ini
163	(t)		A3	and e	res 4,e	outi
164	(u)		A4	and h	res 4,h	
165	**RND**		A5	and l	res 4,l	
166	**INKEY$**		A6	and (hl)	res 4,(hl)	
167	**PI**		A7	and a	res 4,a	
168	**FN**		A8	xor b	res 5,b	ldd
169	**POINT**		A9	xor c	res 5,c	cpd
170	**SCREEN$**		AA	xor d	res 5,d	ind
171	**ATTR**		AB	xor e	res 5,e	outd
172	**AT**		AC	xor h	res 5,h	
173	**TAB**		AD	xor l	res 5,l	
174	**VAL$**		AE	xor (hl)	res 5,(hl)	
175	**CODE**		AF	xor a	res 5,a	
176	**VAL**		B0	or b	res 6,b	ldir
177	**LEN**		B1	or c	res 6,c	cpir
178	**SIN**		B2	or d	res 6,d	inir
179	**COS**		B3	or e	res 6,e	otir
180	**TAN**		B4	or h	res 6,h	
181	**ASN**		B5	or l	res 6,l	
182	**ACS**		B6	or (hl)	res 6,(hl)	
183	**ATN**		B7	or a	res 6,a	
184	**LN**		B8	cp b	res 7,b	lddr
185	**EXP**		B9	cp c	res 7,c	cpdr
186	**INT**		BA	cp d	res 7,d	indr
187	**SQR**		BB	cp e	res 7,e	otdr
188	**SGN**		BC	cp h	res 7,h	
189	**ABS**		BD	cp l	res 7,l	
190	**PEEK**		BE	cp (hl)	res 7,(hl)	

Codice	Caratttere	Hex	Assembler Z80	- dopo CB	- dopo ED
191	**IN**	BF	cp a	res 7,a	
192	**USR**	C0	ret nz	set 0,b	
193	**STR$**	C1	pop bc	set 0,c	
194	**CHR$**	C2	jp nz,NN	set 0,d	
195	**NOT**	C3	jp NN	set 0,e	
196	**BIN**	C4	call nz,NN	set 0,h	
197	**OR**	C5	push bc	set 0,l	
198	**AND**	C6	add a,N	set 0,(hl)	
199	<=	C7	rst 0	set 0,a	
200	>=	C8	ret z	set 1,b	
201	<>	C9	ret	set 1,c	
202	**LINE**	CA	jp z,NN	set 1,d	
203	**THEN**	CB		set 1,e	
204	**TO**	CC	call z,NN	set 1,h	
205	**STEP**	CD	call NN	set 1,l	
206	**DEF FN**	CE	adc a,N	set 1,(hl)	
207	**CAT**	CF	rst 8	set 1,a	
208	**FORMAT**	D0	ret nc	set 2,b	
209	**MOVE**	D1	pop de	set 2,c	
210	**ERASE**	D2	jp nc,NN	set 2,d	
211	**OPEN#**	D3	out (N),a	set 2,e	
212	**CLOSE#**	D4	call nc,NN	set 2,h	
213	**MERGE**	D5	push de	set 2,l	
214	**VERIFY**	D6	sub N	set 2,(hl)	
215	**BEEP**	D7	rst 16	set 2,a	
216	**CIRCLE**	D8	ret c	set 3,b	
217	**INK**	D9	exx	set 3,c	
218	**PAPER**	DA	jp c,NN	set 3,d	
219	**FLASH**	DB	in a,(N)	set 3,e	
220	**BRIGHT**	DC	call c,NN	set 3,h	
221	**INVERSE**	DD	precede le istruzioni relative al registro ix	set 3,l	
222	**OVER**	DE	sbc a,N	set 3,(hl)	
223	**OUT**	DF	rst 24	set 3,a	
224	**LPRINT**	E0	ret po	set 4,b	
225	**LLIST**	E1	pop hl	set 4,c	
226	**STOP**	E2	jp po,NN	set 4,d	
227	**READ**	E3	ex (sp),hl	set 4,e	
228	**DATA**	E4	call po,NN	set 4,h	
229	**RESTORE**	E5	push hl	set 4,l	

Codice	*Carattere*	*Hex*	*Assembler Z80*	*- dopo CB*	*- dopo ED*
230	**NEW**	E6	and N	set 4,(hl)	
231	**BORDER**	E7	rst 32	set 4,a	
232	**CONTINUE**	E8	ret pe	set 5,b	
233	**DIM**	E9	jp (hl)	set 5,c	
234	**REM**	EA	jp pe,NN	set 5,d	
235	**FOR**	EB	ex de,hl	set 5,e	
236	**GO TO**	EC	call pe,NN	set 5,h	
237	**GO SUB**	ED		set 5,l	
238	**INPUT**	EE	xor N	set 5,(hl)	
239	**LOAD**	EF	rst 40	set 5,a	
240	**LIST**	F0	ret p	set 6,b	
241	**LET**	F1	pop af	set 6,c	
242	**PAUSE**	F2	jp p,NN	set 6,d	
243	**NEXT**	F3	di	set 6,e	
244	**POKE**	F4	call p,NN	set 6,h	
245	**PRINT**	F5	push af	set 6,l	
246	**PLOT**	F6	or N	set 6,(hl)	
247	**RUN**	F7	rst 48	set 6,a	
248	**SAVE**	F8	ret m	set 7,b	
249	**RANDOMIZE**	F9	ld sp,hl	set 7,c	
250	**IF**	FA	jp m,NN	set 7,d	
251	**CLS**	FB	ei	set 7,e	
252	**DRAW**	FC	call m,NN	set 7,h	
253	**CLEAR**	FD	precede le istruzioni relative al registro iy	set 7,l	
254	**RETURN**	FE	cp N	set 7,(hl)	
255	**COPY**	FF	rst 56	set 7,a	

APPENDICE B

Messaggi

Ogni volta che il calcolatore si ferma, stampa un messaggio sulla parte bassa dello schermo, che indica perchè si è fermato, o per una ragione normale, o per un errore.

Il messaggio ha un numero di codice o una lettera, che potete usare per cercarne la spiegazione nella tabella che segue, un breve commento in inglese che spiega cosa è successo, il numero della linea e la posizione in detta linea del comando che ha causato l'arresto dell'elaborazione. I comandi in modo immediato sono indicati dal numero di linea 0. Il primo comando nella linea è il numero 1, il secondo è dopo i primi due punti o dopo il **THEN**, ecc.

Anche il comportamento di **CONTINUE** dipende dal messaggio; di solito questo comando salta al numero di linea e al comando specificati nell'ultimo messaggio, ma vi sono delle eccezioni: 0, 9 e D (vedete l'Appendice C).

Tutti i messaggi, le situazioni in cui si possono verificare e una spiegazione del loro significato sono riportati nella seguente tabella; per esempio, l'errore **A Invalid argument** si può verificare con **SQR, IN, ACS** e **ASN** e se consultate l'Appendice C saprete esattamente quali sono gli argomenti non validi.

Codice	*Significato*	*Situazioni*
0	OK Esecuzione terminata con successo, o salto ad un numero di linea più grande di qualunque linea interna al programma. Questo messaggio non modifica la linea comando a cui salta **CONTINUE**.	Qualunque
1	**NEXT without FOR** (**NEXT** senza **FOR**) La variabile di controllo non esiste, non è stata cioè inizializzata da un comando **FOR**, ma vi è una variabile normale che ha lo stesso nome.	**NEXT**
2	**Variable not found** (variabile non trovata) Con una variabile normale, significa che essa viene usata prima che sia stata assegnata in un comando **LET, READ** o **INPUT**, o caricata da nastro, o inizializzata in un comando **FOR**. Per una variabile indicizzata, indica che la variabile viene usata prima di essere dimensionata in un comando **DIM**, o caricata dal nastro.	Qualunque

Codice	*Significato*	*Situazioni*
3	**Subscript wrong** (indice errato) L'indice è più grande della dimensione della matrice, o vi sono un numero errato di indici. Se l'indice è negativo, o più grande di 65535, si otterrà un errore B.	Variabili indicizzate Substringhe
4	**Out of memory** (non c'è più memoria) Non c'è abbastanza spazio nel calcolatore per quello che vorreste fare. Se il calcolatore sembra essere bloccato, in questo stato, può essere necessario cancellare il comando con **DELETE**, e poi cancellare qualche linea di programma, con l'intenzione poi di riinserirla, per creare spazio per poter eseguire una manovra, per esempio **CLEAR**.	**LET, INPUT, FOR, DIM, GO SUB, LOAD, MERGE** Qualche volta durante il calcolo di espressioni
5	**Out of screen** (schermo pieno) Un comando **INPUT** ha cercato di generare più di 23 linee nella parte bassa dello schermo. Succede anche con **PRINT AT 22,...**	**INPUT, PRINT AT**
6	**Number too big** (numero troppo grosso) Dei calcoli hanno dato per risultato un numero maggiore di circa 10^{38}.	Ogni operazione aritmetica
7	**RETURN without GO SUB (RETURN** senza **GO SUB)** Vi è un **RETURN** di troppo rispetto ai **GO SUB**.	**RETURN**
8	**End of file** (fine del file)	Operazioni con Microdrive, ecc.
9	**STOP statement** (comando di **STOP**) È stato incontrato un messaggio di **STOP, CONTINUE** non riprenderà dallo **STOP**, ma dal comando seguente.	**STOP**
A	**Invalid argument** (argomento non valido) L'argomento di una funzione non va bene per qualche ragione.	**SQR, LN, ASN, ACS, USR** (con argomento stringa)
B	**Integer out of range** (intero fuori scala) Quando è necessario un numero intero, l'argomento floating point è arrotondato all'intero più vicino; si avrà questo errore se il risultato non è compreso nell'intervallo conveniente.	**RUN, RANDOMIZE, POKE** **DIM, GO TO, GO SUB, LIST, LLIST, PAUSE,**

Codice	Significato	Situazioni
	Per le matrici, vedere anche l'errore 3.	**PLOT, CHR,$ PEEK, USR** (con argomento numerico)
C	**Nonsense in BASIC** (privo di significato in BASIC) Il testo dell'argomento (stringa) non rappresenta n'espressione valida.	**VAL, VAL$**
D	**BREAK - CONT repeats (BREAK - CONT** ripete) **BREAK** è stato premuto durante qualche operazione con le periferiche. Il comportamento di **CONTINUE** dopo questo messaggio è normale, dato che ripete il comando confrontate con il messaggio L.	**LOAD, SAVE, VERIFY, MERGE, LPRINT, LLIST, COPY.** Anche quando il calcolatore chiede **scroll?** e rispondete **N, SPACE** o **STOP**
E	**Out of DATA** (dati della **DATA** esauriti) Avete cercato di eseguire **READ** quando la lista **DATA** era già esaurita.	**READ**
F	**Invalid file name** (nome di file non valido) **SAVE** è stato usato con un argomento nullo, o più lungo di 10 caratteri.	**SAVE**
G	**No room for line** (non c'è spazio per la linea) Non c'è abbastanza spazio in memoria per la nuova linea di programma.	Quando si introduce una linea nel programma.
H	**STOP in INPUT (STOP** durante **INPUT)** Qualche dato in **INPUT** è stato iniziato con **STOP**, o con **INPUT LINE** è stato premuto **STOP**. Diversamente dal messaggio 9, dopo il messaggio **H, CONTINUE** si comporta normalmente e ripete il comando di **INPUT**.	**INPUT**
I	**FOR without NEXT (FOR** senza **NEXT)** C'era un ciclo **FOR** da non eseguire nessuna volta (per esempio, **FOR n**=1 TO 0), ma non si è potuto trovare il corrispondente comando **NEXT**.	**FOR**
J	**Invalid I/O device** (operazione di I/O non valida)	Operazioni con Microdrive, ecc.

Codice	*Significato*	*Situazioni*
K	**Invalid colour** (colore non valido) Il numero specificato non è un valore consentito.	**INK, PAPER, BORDER, FLASH, BRIGHT, INVERSE, OVER** anche dopo uno dei corrispondenti caratteri di controllo
L	**BREAK into program** (interruzione nel programma) La richiesta di **BREAK** è stata identificata tra due comandi. Il numero di linea presente nel messaggio indica l'ultima linea eseguita, ma **CONTINUE** eseguirà il comando seguente, permettendo anche che venga eseguito un eventuale salto lasciato in sospeso, quindi non ripete alcun comando.	Qualunque
M	**RAMTOP no good (RAMTOP** non valido) Il numero specificato per RAMTOP è troppo alto o troppo basso.	**CLEAR**; a volte **RUN**
N	**Statement lost** (comando perso) Salto a un comando che non esiste più.	**RETURN, NEXT, CONTINUE**
O	**Invalid stream** (flusso non valido)	Operazioni con Microdrive, ecc.
P	**FN without DEF (FN** senza **DEF)** **FN** deve essere preceduto da un **DEF** quando si definisce una funzione definita dall'utente.	**FN**
Q	**Parameter error** (errore di parametro) Numero errato di argomenti, o argomento di tipo sbagliato (stringa invece di numero e viceversa).	**FN**
R	**Tape loading error** (errore di lettura su nastro) L'informazione è stata trovata sul nastro, ma per qualche ragione non può essere letta, oppure non risulta coincidente con quella contenuta in memoria (con **VERIFY**).	**VERIFY, LOAD** o **MERGE**

APPENDICE C

Descrizione riassuntiva dello ZX Spectrum

La tastiera

L'insieme dei caratteri dello ZX Spectrum comprende i *simboli semplici* (lettere, numeri, ecc.) ed i *simboli composti* (istruzioni, messaggi dalla tastiera, ecc.) che vengono introdotti con una sola battuta e non scritti per esteso. Per ottenere tutte queste funzioni e comandi, alcuni tasti presentano fino a sei significati diversi, selezionabili premendo il tasto in questione contemporaneamente a **CAPS SHIFT** o a **SYMBOL SHIFT** e cambiando il *modo* in cui si trova la macchina.

Il modo è indicato da una lettera lampeggiante all'interno del *cursore*, che indica il punto dello schermo ove comparirà il primo carattere battuto.

Il modo K (per keyword, in italiano parola chiave) sostituisce automaticamente il modo L quando il calcolatore attende o un comando o una linea di programma (invece che dei dati in **INPUT**). Questo accade o all'inizio della riga, o subito dopo un **THEN** o dopo : (salvo che nelle stringhe). In questo modo se non usate nessuno **SHIFT**, il primo tasto che premerete potrà essere interpretato o come la keyword scritta sul tasto in bianco, o come una cifra, se è un tasto numerico.

Il modo L (per letters, lettere) si alterna al modo K. In modo L, se non shiftate, il tasto battuto sarà interpretato come il simbolo principale scritto su di esso (nel caso di una lettera, sarà scritta in minuscolo). Sia nel modo K che nel modo L, un tasto premuto insieme a **SYMBOL SHIFT** ritornerà l'elemento scritto in rosso su di esso, ed un tasto numerico con **CAPS SHIFT** sarà invece interpretato come la funzione di controllo scritta in bianco sopra di esso. Con gli altri tasti **CAPS SHIFT** non ha effetto in modo K, mentre in modo L convertirà i caratteri da minuscolo a maiuscolo.

Il modo C (per capitals, maiuscole) è una variante del modo L nella quale tutte le lettere appariranno in maiuscolo: il **CAPS LOCK** cambia il modo L in modo C e viceversa.

Il modo E (per extended, esteso) è usato per ottenere altri caratteri e, soprattutto, altri comandi; si ottiene dopo aver premuto i due tasti **SHIFT** insieme e dura per una sola battuta. In questo modo, una lettera non shiftata darà il carattere o simbolo scritto in verde sopra essa, una shiftata invece quello che è riportato in rosso sotto di esso. Un tasto numerico darà un simbolo se premuto con **SYMBOL SHIFT** altrimenti esso darà una sequenza di controllo del colore.

Il modo G (per graphics, grafica) si ottiene con **GRAPHICS** (premendo **CAPS SHIFT** ed il **9**), e dura fino a quando lo si preme nuovamente, o fino a quando non premete il **9** da solo. Un tasto numerico produrrà un carattere grafico a mosaico predefinito, e tutte le lettere eccetto v, w, x, y e z daranno un carattere grafico definito dall'utente.

Se desiderate ripetere più volte lo stesso carattere, tenetelo premuto per più di 2 o 3 secondi.

Ciò che scrivete sulla tastiera appare nella parte bassa dello schermo così come è, con i caratteri inseriti subito prima del cursore. Il cursore può essere spostato verso sinistra con il **CAPS SHIFT** ed il **5**, o verso destra con il **CAPS SHIFT** e **8**. Il carattere che precede il cursore può essere cancellato con **DELETE** (**CAPS SHIFT** e **0**), e la riga intera con **EDIT** (**CAPS SHIFT** e **1**) seguito da **ENTER**.

Una linea viene, a seconda, eseguita, introdotta nel programma o usata per leggere dei dati, quando viene premuto **ENTER**. Se c'è un errore non viene accettata e appare un **?** lampeggiante vicino ad esso.

Dopo l'inserimento di ogni linea di programma, compare nella parte alta dello schermo un listato del programma. L'ultima linea introdotta è chiamata linea *corrente* ed è indicata dal simbolo **>** dopo il numero di linea; quest'ultimo può essere spostato usando i tasti (**CAPS SHIFT** e **6**) e (**CAPS SHIFT** e **7**). Se viene premuto il tasto **EDIT** (**CAPS SHIFT** e **1**), la linea corrente è riportata sullo schermo in basso e può essere modificata e/o corretta.

Quando la macchina esegue un comando, o un programma, visualizza i risultati in uscita sulla parte alta dello schermo. Vi rimangono fino a quando viene introdotta una linea di programma, o viene comunque premuto **ENTER**, o usate i tasti ↑ o ↓. Nella parte in basso appare un messaggio con un codice (lettera o numero) spiegato nell'appendice B. Rimarrà fino a quando non sarà premuto un qualunque tasto (si è in modo K).

In certi momenti, **CAPS SHIFT** con **SPACE** viene riconosciuto come **BREAK**, e quindi ferma il calcolatore, con un messaggio **D** o **L** ciò avviene

(i) al termine di una linea mentre il programma sta girando, o
(ii) mentre il calcolatore sta usando il registratore o la stampante.

Il video

Lo schermo è formato da 24 linee, ognuna lunga 32 caratteri, ed è diviso in due parti. La parte alta è al massimo di 22 linee, e visualizza o un listato, o un risultato in uscita: se uno di questi ultimi è particolarmente lungo, tanto da raggiungere il bordo inferiore, vi sarà lo scorrimento (scrolling), cioè la prima riga sparirà e tutto il listato si sposterà verso l'alto per far spazio all'ultima riga da stampare. Ogni volta che si riempe il video con un ciclo (nel senso di **FOR...NEXT**) il calcolatore si ferma, con il messaggio **scroll?**, per evitare che delle righe spariscano prima che possano essere esaminate. Se premete **n**, o **SPACE**, o **STOP** il programma si interromperà e apparirà il messaggio **D BREAK - CONT repeats** premendo un altro tasto qualsiasi lo scorrimento continuerà. La parte bassa è usata per l'introduzione dei comandi, delle linee di programma e dei dati, oltre che per la visualizzazione dei messaggi. La sezione bassa dello schermo occupa inizialmente solo due linee, ma si espande verso l'alto se è necessario più spazio. Quando, espandendosi, raggiunge la posizione di stampa corrente, ogni ulteriore espansione farà spostare verso l'alto

anche la parte superiore. Questo fenomeno si può evidenziare usando due colori diversi per **PAPER** e **BORDER** e leggendo qualche dato molto lungo dopo aver stampato diverse linee.

Ogni posizione dello schermo ha i suoi attributi, che determinano quale sia il suo colore di sfondo e quale quello dell'inchiostro, il livello di luminosità e se debba o meno lampeggiare. I colori disponibili sono nero, blu, rosso, magenta, verde, ciano, giallo e bianco.

La cornice dello schermo può assumere uno qualunque dei colori disponibili tramite il comando **BORDER**.

Ogni posizione di carattere è divisa in 8*8 pixels, che "dipinti" individualmente col colore della carta o col colore dell'inchiostro nella propria posizione, permettono di ottenere la grafica ad alta risoluzione.

Gli attributi di una posizione di carattere sono cambiati ogni volta che vi viene stampato un carattere, oppure quando vi viene cambiato anche un solo pixel. Gli attributi assumono i valori dei parametri di stampa settati permanentemente o temporaneamente al momento dell'esecuzione del comando di scrittura. I parametri di stampa sono 6: **PAPER, INK, FLASH, BRIGHT, INVERSE** e **OVER**. Le condizioni all'accensione sono: inchiostro nero su carta bianca, luminosità normale, caratteri stabili, video normale e niente sovraimpressione. I parametri permanenti sono modificati coi comandi **PAPER, INK** ecc., e non vengono alterati fino ad una successiva esecuzione degli stessi comandi. I parametri permanenti per la parte bassa dello schermo hanno sempre il colore del bordo come colore della carta, e colore dell'inchiostro, bianco o nero, sempre contrastante col colore della carta (bordo), luminosità normale, carattere stabile, video normale e niente sovraimpressione.

I parametri temporanei sono settati da **PAPER** e **INK**, ecc. usati come elementi dei comandi **PRINT, LPRINT, INPUT, PLOT, DRAW** e **CIRCLE**, o dai caratteri di controllo equivalenti a **PAPER, INK**, ecc., nel momento in cui vengono stampati sul video. Questi caratteri di controllo devono essere sempre seguiti da un altro byte, che indica il valore del parametro.

I parametri temporanei hanno effetto solo fino alla fine del comando di **PRINT**, ecc., e in un comando di **INPUT** solo fino a quando non vengono richiesti i dati da tastiera; sono quindi rimpiazzati da quelli permanenti.

I parametri per **PAPER** e **INK** possono assumere un valore tra 0 e 9 compreso. I parametri da 0 a 7 indicano i colori secondo la seguente tabella:

0 nero
1 blu
2 rosso
3 magenta
4 verde
5 ciano
6 giallo
7 bianco

Il parametro 8 (trasparenza) significa che quando un carattere viene stampato il colore del parametro deve essere lasciato inalterato; il parametro 9 (contrasto) specifica che il colore in questione (o carta, o inchiostro), deve essere reso o bianco o nero, per contrastare con l'altro colore.

I parametri di **FLASH** e **BRIGHT** possono essere o 0, 1, o 8; 1 significa che il lampeggiamento (**FLASH**), o la luminosità extra (**BRIGHT**) è attiva , 0 che non lo è, 8 (trasparenza) che non è da modificare.

I parametri per **OVER** e **INVERSE** sono analogamente 0 e 1.

OVER 0 fà sì che i nuovi caratteri cancellino quelli vecchi

OVER 1 fà sì che la maschera dei punti del nuovo carattere sia sovrapposta a quella vecchia usando una funzione "exclusive or"

INVERSE 0 fà sì che i caratteri siano stampati come colore d'inchiostro su colore di carta (video normale)

INVERSE 1 fà sì che i caratteri siano stampati con la maschera di "inchiostratura" invertita (video inverso)

Un carattere di controllo **TAB**, mandato al video deve essere seguito da altri due byte, che indicano il **TAB** desiderato (col byte meno significativo per primo). Il numero specificato e ridotto in modulo 32, e quindi vengono stampati abbastanza spazi per spostare il cursore fino al numero di colonna ottenuto.

Quando è ricevuto il carattere di controllo virgola, vengono stampati tutti gli spazi sufficienti (da un minimo di 1) per spostare la posizione di stampa alle colonne 0 o 16.

Quando viene premuto **ENTER**, la posizione di stampa viene spostata alla linea successiva.

La stampante

L'uscita per la stampante ZX printer viene effettuata tramite un buffer di 32 caratteri, che rappresenta una linea di stampa che viene trasferita per intero quando

(i) il buffer è pieno, ed è quindi necessario iniziare a scrivere sulla nuova linea,
(ii) quando viene premuto **ENTER**,
(iii) al termine del programma, se c'è ancora qualcosa nel buffer che non è stato stampato,
(iv) quando un **TAB**, una virgola, un carattere di controllo spostano la posizione di stampa su una nuova linea.

I **TAB** e le virgole mandano alla stampante gli spazi allo stesso modo che sullo schermo.

Inoltre, il comando **AT** sposta la posizione di stampa nel buffer usando il numero

di colonna, ma ignora il numero di linea, quindi non produce mai di per sè la stampa di una linea.

I comandi **INVERSE** e **OVER** hanno sulla stampante lo stesso effetto che hanno sul video, diversamente dai comandi **PAPER, FLASH, INK** e **BRIGHT**, che non ne hanno alcuno.

La stampante può essere fermata con un messaggio di tipo B premendo il tasto **BREAK**

Se la stampante non è collegata quando è richiesto l'output su di essa, i dati in uscita vengono semplicemente persi.

Il BASIC

I numeri sono memorizzati con 9 o 10 cifre significative. Il numero più grosso che può essere memorizzato è circa 10^{38}, ed il più piccolo, positivo, è circa $4*10^{-39}$.

Nello ZX Spectrum, un numero è memorizzato in forma binaria floating point, con un byte di esponente "e" ($1<=e<=255$) e quattro bytes di mantissa "m" ($1/2<=m<1$), che rappresentano $m*2^{e-128}$.

Numeri interi piccoli hanno una speciale rappresentazione per cui il primo byte è 0, il secondo è un byte di segno, 0 o FFh, il terzo e il quarto rappresentano l'intero nella forma in complemento a due, col byte meno significativo per primo, e il quinto byte è 0.

Le variabili numeriche hanno un nome di lunghezza a piacere, che deve iniziare con una lettera, e che può continuare con lettere e cifre. Gli spazi ed i controlli per il colore vengono ignorati e tutte le lettere vengono considerate come minuscole.

I nomi delle variabili di controllo per i cicli **FOR-NEXT** devono essere formati da una sola lettera, così come le matrici numeriche. Queste ultime possono avere nomi uguali a variabili semplici senza che si crei confusione, possono avere quante dimensioni si vuole, e ogni dimensione può essere grande a piacere. La base degli indici è 1, vale a dire che, a differenza di molti calcolatori, se si dimensiona una matrice di 10, non esiste la posizione 0, e la matrice è quindi realmente composta da dieci elementi.

Non vi è alcuna restrizione alla lunghezza delle stringhe. Il nome di una variabile stringa deve essere dato da una sola lettera seguita da $.

Le matrici di stringhe non possono avere lo stesso nome di una stringa semplice, possono avere quante dimensioni si vuole. Tutte le stringhe di una certa matrice hanno una stessa lunghezza prefissata, che è specificata con un'extra dimensione finale nel comando di **DIM**; usando questa dimensione extra anche nei comandi di lettura, è possibile leggere un dato carattere di una certa stringa della matrice, quindi le matrici di stringhe si comportano anche come matrici di caratteri. Anche per le matrici di stringhe gli indici hanno base 1.

Suddivisione di stringhe (slicing): usando i divisori, si possono creare delle sottostringhe, che diventano a loro volta stringhe normali. Un divisore può essere

(i) vuoto

o

(ii) un'espressione numerica

o

(iii) espressione numerica opzionale **TO** espressione numerica opzionale

ed è usato per indicare una sottostringa nei formati

(a) espressione stringa (divisore)

(b) nome di matrice stringa (indice,..., indice, divisore)

che significa lo stesso che

nome di matrice stringa (indice,..., indice) (divisore)

Nel caso (a), supponete che l'espressione stringa abbia il valore s$.

Se il divisore è vuoto, il risultato è s$, che viene considerato come sottostringa di se stesso.

Se il divisore è un'espressione numerica con valore n, il risultato è l'ennesimo carattere della stringa s$, cioè una sottostringa di lunghezza 1.

Se il divisore è nella forma (iii), supponete che la prima espressione numerica abbia valore m (se non è specificato assume automaticamente il valore 1) e la seconda n (se non specificata assume la lunghezza di s$).

Se 1< = m < = n< = *lunghezza di s$*, il risultato è una sottostringa di s$ che inizia con l'emmesimo carattere e termina con l'ennesimo. Se 0 < = n < = m il risultato è una stringa vuota, atrimenti si ha un errore di tipo 3.

La divisione di stringa viene eseguita prima del calcolo di funzioni ed espressioni, a meno che le parentesi non indichino il contrario.

Le sottostringhe possono essere assegnate (vedere **LET**).

Se le virgolette devono far parte di una stringa, come carattere, devono essere raddoppiate.

Se non fosse ancora chiara la differenza tra stringhe e sottostringhe: una stringa è una qualunque variabile stringa, o una qualunque espressione stringa, comunque ottenuta e di lunghezza anche variabile; una sottostringa è il risultato di un'operazione su stringhe contenente dei divisori, e di lunghezza prestabilita.

Funzioni

Gli argomenti di una funzione non hanno bisogno di parentesi se l'argomento è una costante o una variabile anche indicizzata, o il risultato di uno slicing.

Funzione	*Tipo di argomento* (x)	*Risultato*
ABS	numero	Valore assoluto.
ACS	numero	Arcoseno in radianti. Se x non è compreso tra —1 e +1, si ha l'errore A
AND	operazioni binarie, l'operando di destra deve essere sempre un numero.	
	Con argomento di sinistra numerico:	A **AND** B = { A se B <> 0 ; 0 se B=0
	Con argomento di sinistra stringa:	A$ **AND** B = { A$ se B <> 0 ; "" se B=0
ASN	numero	Arcoseno in radianti. Se x non è compreso tra —1 e +1 si ha l'errore A
ATN	numero	Arcotangente in radianti
ATTR	due argomenti, x e y entrambi numeri tra parentesi	Un numero la cui forma binaria rispecchia gli attributi della linea x colonna y del video. Bit 7 (più significativo) è 1 per lampeggiamento, 0 per carattere stabile. Bit 6 è 1 per luminosità extra, 0 per normale. Bit 5, 4 e 3 sono per il colore della carta, i rimanenti per il colore dell'inchiostro. Se x non è 0 < =x < = 23 e y non è 0 < =y < =31 si ha l'errore B.
BIN	binario, una serie di 0 ed 1	Non è realmente una funzione, ma una rappresentazione alternativa dei numeri: **BIN** viene seguito da una sequenza di 0 ed 1, che sono la rappresentazione binaria del numero che si vuole scrivere.
CHR$	numero	Carattere di codice x, arrotondato all'intero più vicino
CODE	stringa	Il codice del primo carattere nella stringa, oppure 0 se la stringa è vuota
COS	numero (in radianti)	Coseno di x
EXP	numero	n^e

Funzione	*Tipo di argomento*	*Risultato*
FN		**FN** seguito da una lettera è una funzione definita dall'utente (vedere **DEF**). Le parentesi devono essere usate anche se non vi è argomento.
IN	numero	Risultato dell'input a livello processore dalla porta x (0 < =x < = FFFFh) (carica la coppia di registri bc con x ed esegue le istruzioni in linguaggio macchina in a(c))
INKEY$	nessuno	Legge la tastiera. Il risultato è il carattere rappresentato (in modo C o L) dal tasto premuto, se ce n'è premuto uno nel momento preciso in cui la funzione viene eseguita, altrimenti ritorna una stringa nulla.
INT	numero	Parte intera, sempre arrotondata per difetto
LEN	stringa	Lunghezza della stringa
LN	numero	Logaritmo naturale di base e. Errore A se x < =0
NOT	numero	Se x < > 0 è 0, se x=0 è 1. **NOT** ha priorità 4.
OR	operazione binaria, entrambi gli operandi numerici	$A \text{ OR } B = \begin{cases} 1 \text{ se } b <> 0 \\ A \text{ se } B=0 \end{cases}$ **OR** ha priorità 2
PEEK	numero	Valore del byte di memoria di indirizzo x arrotondato all'intero più vicino. Se x non è compreso tra 0 e 65535 si ha un errore B
PI	nessuno	Ritorna π (3.14159265...)
POINT	due argomenti, x e y, entrambi numeri racchiusi tra parentesi	Ritorna 1 se il pixel (x,y) è del colore dell'inchiostro, 0 se del colore della carta. Se x non è 0 < =x < =255 e se y non è 0 < =y < =175 si ha errore B
RND	nessuno	Il successivo numero pseudocasuale di una sequenza generata prendendo le potenze di 75 in modulo 65537, sottraendo 1 e dividendo per 65536. 0 < = y < 1

Funzione	*Tipo di argomento*	*Risultato*
SCREEN$	due argomenti, x e y entrambi numeri racchiusi tra parentesi	Il carattere che appare sul video, sia in normale che in campo inverso, alla linea x colonna y. Se il carattere non viene riconosciuto ritorna una stringa nulla (per esempio, se è stato disegnato con la **POINT**, o la **PLOT**). Se non è $0 <= x <= 23$ e $0 <= y <= 31$ si ha l'errore B
SGN	numero	Segno di x —1 se negativo, 0 per lo zero e 1 se è positivo
SIN	numero (in radianti)	Seno di x
SQR	numero	Radice quadrata. Se $x < 0$ si ha errore A
STR$	numero	La stringa di caratteri che apparirebbe sul video se x fosse stampato
TAN	numero in radianti	Tangente di x
USR	numero	Chiama la routine in linguaggio macchina che inizia con l'indirizzo x, e ritorna il valore della coppia di registri *bc* al termine della routine.
USR	stringa	Indirizzo della maschera di bit per il carattere grafico definito dall'utente corrispondente a x; se A non è una lettera singola tra A ed U o un carattere grafico definito dall'utente, si ha l'errore A
VAL	stringa	Calcola x (senza gli apici che lo delimitano) come un'espressione numerica. Se x contiene un errore di sintassi o dà un valore di stringa, si ha un errore C. Sono possibili altri errori a seconda dell'espressione.
VAL$	stringa	Calcola x (senza gli apici che lo delimitano) come un'espressione stringa. Se x contiene un errore di sintassi o dà un valore numerico, si ha errore C. Sono possibili altri errori a seconda dell'espressione.
—	numero	Negazione

Le seguenti sono espressioni binarie:

+ Addizione (sui numeri) e concatenazione (sulle stringhe)
− Sottrazione
* Moltiplicazione

/	Divisione	
↑	Elevamento a potenza. Se l'operando di sinistra è negativo si ha errore B	
=	Uguale a	Entrambi gli operandi devono essere dello stesso tipo. Se la relazione risulta vera, il risultato è 1, diversamente è 0
>	Maggiore di	
<	Minore di	
<=	Monore o uguale	
>=	Maggiore o uguale	
<>	Diverso da	

Le funzioni e le operazioni hanno le seguenti priorità:

Operazioni	*Priorità*
Indici e slicing	12
Tutte le funzioni, salvo **NOT** ed il meno di negazione	11
Elevamento a potenza	10
Meno di negazione (—), usato solo per negare qualcosa	9
*, /	8
+, − (meno usato per sottrarre una cosa da un'altra)	6
=, >, <, <=, > =, <>	5
NOT	4
AND	3
OR	2

Comandi

Nella tavola che segue,

α	rappresenta una singola lettera
v	rappresenta una variabile
x,y,z	rappresentano espressioni numeriche
m,n	rappresentano espressioni numeriche arrotondate all'intero più vicino
e	rappresenta un'espressione
f	rappresenta un'espressione che abbia valore di stringa
s	rappresenta una sequenza di comandi separati da due punti
c	rappresenta una sequenza di comandi di colore, ognuno terminato da virgola (,) o punto e virgola (;). Ogni comando di colore può essere nella forma di **PAPER, INK, FLASH, BRIGHT, INVERSE** o **OVER**.

Notate che, a parte che per il numero di linea all'inizio di un comando, si possono usare dovunque espressioni di qualunque tipo o complessità.

Tutti i comandi, salvo **INPUT, DEF** e **DATA** possono essere usati in modo diretto come nei programmi, anche se possono essere più utili in un modo che nell'altro. Un comando in modo immediato o una linea di programma possono avere diversi elementi, separati dai due punti. Non vi sono particolari restrizioni su dove in una linea possa essere inserito un certo comando, ma si faccia attenzione a **IF** e **REM**, che condiziona il resto della linea.

BEEP x, y	Produce una nota attraverso l'altoparlante di lunghezza x secondi e di y semitoni sopra il DO centrale, o sotto se y è negativo. Sono permessi i valori frazionari.
BORDER m	Determina il colore per il bordo dello schermo, che è anche il colore della carta per la parte bassa dello schermo. Se m non è compreso tra 0 e 7 si ha un errore K.
BRIGHT	Determina la luminosità dei caratteri stampati dal momento dell'esecuzione del comando in poi: n=0 per luminosità normale, 1 per extraluminosità e 8 per trasparenza. Se n non è 0, 1 o 8, si ha un errore K.
CAT	Non funziona senza Microdrive, ecc.
CIRCLE x, y, z	Disegna una circonferenza di centro x, y e di raggio z.
CLEAR	Cancella tutte le variabili, liberando lo spazio che occupano. Esegue **RESTORE** e **CLS**, resetta la posizione di **PLOT**, all'angolo sinistro in basso dello schermo, e pulisce lo stack della **GO SUB**.
CLEAR n	Come **CLEAR**, ma in più, se possibile, cambia la variabile di sistema **RAMTOP** a n e alloca il nuovo stack per la **GO SUB** a partire da quella posizione.
CLOSE#	Non funziona senza Microdrive, ecc.
CLS	Cancella lo schermo, cancella il display file.
CONTINUE	Continua il programma, a partire da dove si era fermato con un qualunque messaggio diverso da 0. Se il messaggio era 9 o L, continua con il comando seguente all'ultimo eseguito (tenendo conto dei salti eventualmente in sospeso); altrimenti ripete il comando che aveva causato l'interruzione. Se l'ultimo messaggio era in una linea di comando (IMMEDIATO), **CONTINUE** cercherà di continuarla e cadrà in un ciclo senza fine se l'errore era in 0:1, darà un messaggio 0 se era in 0:2 e darà l'errore N se era in 0:3 o maggiore. **CONTINUE** sulla tastiera è abbreviato **CONT**.
COPY	Manda una copia delle prime 22 linee di schermo alla stampante, se è collegata, altrimenti non fa nulla; comunque **COPY** non può essere usata per stampare il listing automatico, dato che viene cancellato ogni volta che si introduce un comando. Se viene premuto **BREAK** durante **COPY** si ha un messaggio D.

DATA e_1, e_2, e_3 ...	Parte di una lista **DATA** che deve essere in un programma.
DEF FN α (α,...,α_k) = e	Funzione definita dall'utente; deve essere in un programma. Ogni α è una singola lettera o una singola lettera seguita da '$' per argomento o risultato stringa. Se non vi sono argomenti, prende la forma **DEF FN** α () = e.
DELETE f	Non funziona senza Microdrive, ecc.
DIM $\alpha(n_1,...,n_k)$	Cancella ogni matriće di nome α e inizializza una matrice α di k dimensioni $n_1,...,n_k$ e inizializza tutti i valori a 0.
DIM $\alpha\$(n_1,...,n_k)$	Cancella ogni matrice o stringa col nome α\$ e inizializza una matrice di caratteri di k dimensioni, $n_1,...,n_k$ e inizializza tutti i valori a " ". Può essere considerata come una matrice di stringhe di lunghezza fissa n_k, di k-1 dimensioni $n_1,...,n_{k-1}$. Se non vi è abbastanza spazio in memoria per la matrice, si ha un errore 4. Una matrice non è definita fino a quando non è dimensionata in un comando **DIM**
DRAW x, y	**DRAW** x, y, 0
DRAW x, y, z	Disegna una linea spostandosi orizzontalmente di x, verticalmente di y, relativamente alla corrente posizione di **PLOT**, e ruotando intorno ad un angolo z. Se z = 0 disegna una linea retta; se esce dallo schermo si ha un errore B.
ERASE	Non funziona senza Microdrive, ecc.
FLASH	Definisce se i caratteri stampati successivamente al comando devono essere lampeggianti o stabili; n=0 per stabili, n=1 per lampeggianti, n=8 per nessun cambiamento.
FOR α=x **TO** y	**FOR** α=x **TO** y **STEP** 1
FOR α=x **TO** y **STEP** z	Cancella ogni variabile semplice di nome α e inizializza una variabile di controllo α con valore x, limite y e incremento z e, come indirizzo di ritorno, quello del comando dopo il **FOR**. Controlla se il valore iniziale è maggiore (step α>=0) o minore (step <=0) del limite, e se è così, salta al comando **NEXT** α, dando errore 1 se non lo trova. Vedere **NEXT**. Se non vi è spazio per la variabile di controllo, si verifica un errore 4.
FORMAT f	Non funziona senza Microdrive, ecc.

GO SUB n

Inserisce il numero di linea dove si trova il comando **GO SUB** in uno stack della **GO SUB**, e quindi funziona come **GO TO** n. Se non vi sono abbastanza **RETURN** si può verificare un errore di tipo 4.

GO TO n

Salta alla linea n o, se non c'è, alla prima linea di numero più grande di n.

IF x THEN s

Se x è vero, non 0, s viene eseguito. Notate che s comprende tutti i comandi fino al termine della linea. La forma **IF** x **THEN** numero di linea non è consentita.

INK n

Determina il colore dell'inchiostro di tutti i caratteri stampati dopo il comando. n può essere compreso tra 0 e 7 per un colore, 8 per la trasparenza e 9 per il contrasto. Vedere Appendice C, parte 1, schermo e colori. Se n non è compreso tra 0 e 9 si ha un errore K.

INPUT ...

Al posto dei puntini si ha una sequenza di elementi di **INPUT**, separati come nella **PRINT** da virgole, punti e virgola o apostrofi. Un elemento di **INPUT** può essere:

(i) Ogni elemento di **PRINT** che non inizi con una lettera, che verrà stampata (le variabili di cui si vuole stampare il valore dovranno essere racchiuse tra parentesi)

(ii) Un nome di variabile da leggere

(iii) **LINE**, seguito da un nome di variabile tipo stringa.

Gli elementi delle **PRINT** e i separatori nel caso (i), sono trattati esattamente come nella **PRINT**, ma tutto viene stampato nella parte bassa dello schermo. Nel caso (ii), il calcolatore si ferma e attende che venga scritta da tastiera un'espressione o una variabile. Stampa le lettere battute al solito modo, e se si commette un errore, si ottiene il solito ? lampeggiante. Per espressioni di tipo stringa, il buffer di **INPUT** è inizializzato per contenere due apici, che possono essere cancellati se necessario. Se il primo carattere in input è **STOP**, il programma si ferma con un messaggio H. (iii) è simile al (ii), salvo che l'input è trattato come una stringa senza virgolette e lo **STOP** non funziona; per ottenere lo **STOP** bisogna premere al suo posto

INVERSE n	Controlla l'inversione dei caratteri stampati dopo l'esecuzione del comando. Se n=0, i caratteri sono stampati in modo normale, cioè colore di inchiostro su colore di carta, mentre se n=1 i caratteri vengono stampati in video inverso, cioè colore della carta su colore dell'inchiostro. Notate che viene invertita la maschera dei pixel e non vengono scambiati i colori di **PAPER** e **BORDER**. Vedere anche Appendice C: parte 1, sullo schermo. Se n non è 0 o 1, si ha errore K.
LET v=e	Assegna il valore dell'espressione e alla variabile v. **LET** non può essere omesso. Una variabile semplice non è definita fino a quando non è assegnata in un comando **LET, READ** o **INPUT**. Se v è una variabile stringa indicizzata o una sottostringa, allora l'assegnazione è a lunghezza fissa e il valore "e" è troncato se troppo lungo, e colmato di spazi sulla destra se troppo corto.
LIST	**LIST** 0
LIST n	Lista il programma nella parte superiore dello schermo, iniziando con la prima linea il cui numero è almeno n, e rendendo n la linea corrente.
LLIST	**LLIST** 0
LLIST n	Uguale a **LIST**, ma sulla stampante.
LOAD f	Carica programma e variabili.
LOAD f **DATA** ()	Carica una matrice numerica.
LOAD f **DATA** $()	Carica una matrice stringa.
LOAD f **CODE** m, n	Carica al massimo n bytes, iniziando dall'indirizzo m.
LOAD f **CODE** m	Carica i bytes a cominciare dall'indirizzo n.
LOAD f **CODE**	Carica i bytes all'indirizzo dove erano stati prelevati.
LOAD f **SCREEN$**	Equivale a **LOAD** f **CODE** 16384,6912. Cerca il file del tipo byte per video sul nastro e lo carica, cancellandone dalla memoria versioni precedenti. Vedere capitolo 29.
LPRINT	Come **PRINT** ma sulla stampante.
MERGE f	Come **LOAD** f, ma non cancella il vecchio programma e le vecchie variabili, a meno che non sia necessario per fare spazio a nuove linee omonime.
MOVE f_1, f_2	Non funziona senza il microdrive, ecc.
NEW	Fa ripartire il sistema BASIC da capo, cancellando programma e variabili e usando la memoria fino e compreso all'indirizzo puntato da RAMBOT, e con-

	serva le variabili di sistema UDG, P RAMT, RASP e PIP.
NEXT α	(i) Trova la variabile di controllo α (ii) Somma il suo incremento al suo valore (iii) Se l'incremento è maggiore o uguale a 0 e il valore è minore del limite, o se l'incremento è minore di 0 e il valore è maggiore del limite, salta al comando di ripetizione (**FOR**). Ritorna errore 2 se non compare la variabile α, errore 1 se quella esistente non è una variabile di controllo.
OPEN #	Non funziona senza il microdrive, ecc.
OUT m, n	Mette in uscita sulla porta m il byte n a livello processore. (Carica la coppia di registri bc con m, il registro a con n, esegue l'istruzione in linguaggio assembler: out (c),a). Deve essere 0<=m<=65535, —255<=n<=255, altrimenti si ha un errore B.
OVER n	Controlla la sovraimpressione per i caratteri stampati dopo l'esecuzione del comando. Se n=0, i caratteri stampati cancellano i caratteri precedentemente scritti; se n=1, i nuovi caratteri sono stampati sopra quelli vecchi, e si ha il colore dell'inchiostro dove uno dei due ma non entrambi hanno il colore dell'inchiostro, e il colore della carta dove entrambi hanno il colore della carta o il colore dell'inchiostro. Vedere Appendice C, parte 1, sullo schermo. Se n non è 0 o 1, si ha un errore K.
PAPER n	Analogo a **INK**, ma controlla il colore della carta di sfondo.
PAUSE n	Ferma il calcolatore mostrando il video per n scansioni (50 scansioni per secondo), o fino a quando viene premuto un tasto dopo l'inizio della pausa. Si deve avere 0<=n<=65535, altrimenti si ha errore B. Se n=0 la pausa non viene regolata, ma dura fino a quando non viene premuto un tasto. Ogni pausa viene troncata se viene premuto un tasto dopo il suo inizio.
PLOT c; m,n	Disegna una "macchia d'inchiostro" (soggetta a **OVER** e **INVERSE**) al pixel(|m|, |n|) e sposta la posizione di **PLOT**. A meno che gli elementi di colore c non specifichino diversamente, il colore dell'inchiostro della posizione di carattere in cui viene stampato il pixel è modificato e diventa il

colore dell'inchiostro corrente, il colore della carta, i lampeggiamenti e la luminosità non vengono modificati. Deve essere 0 < =m <=255, 0 <=n <=175, altrimenti si ha errore B.

POKE m, n

Scrive il valore n nel byte indirizzo m. Deve essere 0 <= m <=65535, 0 <= |n| <=255, altrimenti si ha errore B.

PRINT ...

I puntini sono una sequenza di elementi di **PRINT**, separati da virgole, punti e virgola o apostrofi, e vengono scritti nel display file per essere mostrati sul video.

Un punto e virgola tra due elementi non ha effetto, li fa stampare uno di seguito all'altro, una virgola fa stampare il carattere di controllo della virgola e un apostrofo fa stampare il carattere di **ENTER**. Al termine di un comando **PRINT** viene stampato un carattere di **ENTER**, a meno che il comando stesso non termini con virgola, punto e virgola o apostrofo.
Un elemento della **PRINT** può essere:
(i) vuoto, per esempio niente
(ii) un'espressione numerica.

Innanzitutto viene stampato il segno meno se il numero è negativo, e supponendo che x sia il modulo del valore, se x <=10^{-5} oppure x >(=10^{13}, viene stampato usando la notazione scientifica. La mantissa può avere fino a 8 cifre, senza 0, immediatamente seguenti il punto (cioè normalizzata).
Il punto decimale, se si tratta solo di una cifra, segue la prima. La parte di esponente consiste in E, seguita da + o —, seguito da uno o due cifre. Diversamente x viene stampato nella notazione normale, fino a 8 cifre significative e senza 0 di riempimento dopo il punto decimale. Un punto decimale all'inizio deve essere sempre seguito da uno 0; per esempio .03 e 0.3 sono stampati così. 0 viene stampato come una singola cifra 0.
(iii) un'espressione stringa

Le parole chiave nella stringa sono espanse con uno spazio prima o dopo. I caratteri di controllo hanno il loro effetto di controllo, i caratteri non riconosciuti sono stampati come ? .
(iv) **AT** m, n

Stampa un carattere di controllo **AT** seguito da un

	byte m per il numero di linea, e da n per il numero di colonna. (v) **TAB** n Stampa un carattere di controllo **TAB** seguito da n in due bytes (il meno significativo per primo), che indicano lo stop del **TAB**. (vi) Un elemento di colore che è nella forma di un comando **PAPER, INK, FLASH, BRIGHT, INVERSE** o **OVER**.
RANDOMIZE	**RANDOMIZE** 0
RANDOMIZE n	Altera la variabile di controllo chiamata SEED, usata per generare il successivo valore di **RND**. Se n < > SEED assume il valore n; se n=0 gli viene attribuito il valore di un'altra variabile di sistema, chiamata FRAMES, che conta i quadri mostrati sullo schermo dall'accensione, e di solito si comporta come se fosse casuale. **RANDOMIZE** è scritto **RAND** sulla tastiera. Se n non è compreso tra 0 e 65535 si ha errore B.
READ $v_1, v_2, ..., v_k$	Assegna le variabili usando espressioni successive della lista **DATA**. Se l'espressione è di tipo sbagliato, si ha errore di tipo C. Se vi sono ancora variabili da assegnare quando la lista **DATA** è terminata, si ha errore E.
REM ...	Non ha effetto. '...' sono una qualunque sequenza di caratteri, escluso **ENTER**. Questa sequenza può includere i due punti, quindi non sono possibili altri comandi dopo un comando **REM**.
RESTORE	**RESTORE** 0
RESTORE n	Resetta il puntatore dei dati al primo comando **DATA**, con numero di linea almeno n; il **READ** seguente comincerà a leggere da qui.
RETURN	Prende un indirizzo di riferimento dallo stack della **GO SUB** e salta alla linea seguente; se non c'è, si ha errore 7; questo indica che vi deve essere qualche errore nel programma, dato che i **RETURN** non sono propriamente bilanciati dai **GO SUB**.
RUN	**RUN** 0
RUN n	**CLEAR** e **GO TO** n
SAVE f	Memorizza il programma e le variabili.
SAVE f **LINE**	Memorizza il programma e le variabili in modo tale che, quando il programma è caricato, sia eseguito un salto automatico all'inizio.

SAVE f **LINE** m	Memorizza il programma e le variabili in modo tale che, quando è caricato, venga eseguito un salto automatico alla linea m.
SAVE f **DATA** ()	Memorizza una matrice numerica.
SAVE f **DATA** $()	Memorizza una matrice di caratteri.
SAVE f **CODE** m, n	Memorizza n bytes a partire dall'indirizzo m.
SAVE f **SCREEN$**	**SAVE** f **CODE** 16384,6912. Memorizza un'informazione su cassetta dando il nome f; se f è vuoto o ha più di 10 caratteri, si ha un errore F. Per tutte queste **SAVE** consultare il capitolo 29.
STOP	Ferma il programma con un messaggio 9. **CONTINUE** farà eseguire il comando successivo.
VERIFY	Come **LOAD**, solo che non carica i dati nella RAM, ma li confronta con quelli già presenti. Ritorna un errore R se rivela anche una sola differenza. Da usare ogni volta che si esegue una **SAVE** per verificare che l'informazione sia stata memorizzata correttamente, salvo dopo **SAVE** f **SCREEN$**.

APPENDICE D

Esempi di programmi

In questa Appendice sono riportati alcuni programmi provati sullo Spectrum; essi sono commentati in modo da mettere in risalto le grandi possibilità del calcolatore. I listati e gli esempi sono ottenuti con la stampante ZX PRINTER collegata allo Spectrum. In alcuni casi si è usata la stampante per ottenere con il comando COPY il contenuto del video. I programmi si mandano in esecuzione con il comando RUN. Se volete memorizzarli su nastro in modo che dopo il caricamento partano da soli, dovete usare *in modo immediato* il comando:

SAVE nome-programma LINE 9000

ricordando però, che quando un programma ha l'autostart è meglio che inizi inviando al video un messaggio che chiede di fermare il nastro (utile soprattutto se la cassetta contiene più programmi); dovete aggiungere le seguenti linee:

```
9000 CLS:PRINT "FERMA IL NASTRO"; ' ' "Premi un tasto per iniziare":
     PAUSE 0
9010 RUN
```

infatti nessun programma ha la linea 9000. Con il comando SAVE usato tutti i programmi partono da 9000; con l'aggiunta delle linee 9000 e 9010 inviano i messaggi necessari sul video e poi con RUN partono dalla prima istruzione.

CHEGIORNO

Questo programma chiede come dati di ingresso: giorno, mese e anno, e fornisce come risultato il giorno della settimana corrispondente. Funziona solo per anni del ventesimo secolo, cioè dal 1901 al 2000.

```
  10 REM da data a giorno settim
ana
  20 DIM d$(7,10): REM giorni se
ttimana
  30 FOR n=1 TO 7: READ d$(n): N
EXT n
  40 DIM m(12): REM lunghezze me
si
  50 FOR n=1 TO 12: READ m(n): N
EXT n
 100 REM input data
 110 INPUT "giorno? ";g
 120 INPUT "mese? ";m
 130 INPUT "anno (solo 20esimo s
ecolo)? ";a
```

```
140 IF a<1901 THEN PRINT "20esimo secolo inizia anno 1901": GO TO 100
150 IF a>2000 THEN PRINT "20esimo secolo termina anno 2000": GO TO 100
160 IF m<1 THEN GO TO 210
170 IF m>12 THEN GO TO 210
180 IF a/4-INT (a/4)=0 THEN LET m(2)=29: REM anno bisestile
190 IF g>m(m) THEN PRINT "Questo mese ha solo ";m(m);" giorni.": GO TO 500
200 IF g>0 THEN GO TO 300
210 PRINT "Non scherzare! Dammi una data valida.": GO TO 500
300 REM calcola numero giorni corispondenti alla data da inizio del secolo
310 LET y=a-1901
320 LET i=365*y+INT (y/4): REM numero giorni da inizio anno
330 FOR n=1 TO m-1: REM somma a i mesi precedenti
340 LET i=i+m(n): NEXT n
350 LET i=i+g
400 REM conversione in giorno della settimana
410 LET i=i-7*INT (i/7)+1
420 PRINT g;"/";m;"/";a
430 FOR n=10 TO 6 STEP -1: REM toglie spazi inutili
440 IF d$(i,n)<>" " THEN GO TO 460
450 NEXT n
460 LET e$=d$(i, TO n)
470 PRINT "Il giorno e' ";e$
500 LET m(2)=28: REM rimette a posto febbraio
510 INPUT "Ancora (s/n)? ";a$
520 IF a$="n" OR a$="N" THEN GO TO 540
530 GO TO 100
1000 REM giorni settimana
1010 DATA "lunedi'","martedi'","mercoledi'","giovedi'"
1020 DATA "venerdi'","sabato","domenica"
1100 REM numero giorni dei mesi
1110 DATA 31,28,31,30,31,30
1120 DATA 31,31,30,31,30,31
```

Nelle linee da 1000 a 1120 sono usate le frasi DATA per memorizzare le descrizioni dei giorni della settimana e il numero dei giorni di ogni mese. Le 7 descrizioni e i 12 numeri formano il magazzino di dati nel quale vanno a prelevare le istruzioni READ del programma. All'inizio il puntatore che serve per gestire l'insieme dei dati si trova prima di "lunedì". Le linee da 10 a 50 provvedono a trasferire rispettivamente:

1) nel vettore di stringhe D$, formato da 7 stringhe tutte di 10 caratteri, le descrizioni dei giorni,

2) nel vettore di numeri M, formato da 12 elementi, i numeri dei giorni dei mesi. L'assegnazione dei dati alle variabili avviene solo all'inizio del programma e quindi non è necessario fare uso della RESTORE per riposizionare il puntatore all'inizio dell'insieme di dati.

Dalla linea 100 alla 210 si ha il colloquio per l'introduzione dei dati con controllo degli stessi ed eventuale modifica del numero di giorni di febbraio se l'anno è bisestile.

Nelle linee da 300 a 470 viene sviluppato l'algoritmo di calcolo per trovare la corrispondenza tra data e giorno settimana. Viene calcolato il numero di giorni trascorsi dal 1/1/1901, che corrispondeva a un venerdì. Dal numero dei giorni totale viene tolto il numero di giorni corrispondente alle settimane complete, poi viene aggiunto 1 e questo numero serve come indice per trovare il giorno della settimana.

Alla linea 500 viene risistemato il numero di giorni di febbraio a 28. Alla linea 510 viene chiesto se si vuole usare ancora il programma, e, in caso di risposta affermativa si torna alla linea 100. In caso di risposta negativa il programma prosegue, ma, non incontrando istruzioni eseguibili, si ferma.

Risultati ottenuti con la COPY dopo aver risposto "n" alla domanda della linea 510.

```
1/1/1983
Il giorno e' sabato
31/12/1983
Il giorno e' sabato
1/2/1983
Il giorno e' martedi'
28/2/1983
Il giorno e' lunedi'
1/1/1999
Il giorno e' venerdi'
31/12/1999
Il giorno e' venerdi'
25/12/1982
Il giorno e' sabato
31/12/1982
Il giorno e' venerdi'
1/3/1983
Il giorno e' martedi'
```

MISURE

Questo programma è molto semplice, ma mostra come sia comodo ricorrere alla tecnica dei sottoprogrammi. Inizialmente viene mostrata sul video la corrispondenza tra iarde, piedi, pollici e centimetri. Poi, alla linea 10, viene chiesta una misura espressa in iarde, piedi e pollici. Segue alla linea 40 la chiamata del sottoprogramma situato alla linea 2000 per stampare i dati introdotti. Alla linea 50 viene stampato

"=" e non va a capo (; finale); poi i dati di input vengono normalizzati e stampati nuovamente (linee 70 e 80). La linea 90 serve per mandare a capo. La linea 95 richiama il sottoprogramma situato in 3000 per trasformare le misure inglesi in misure metrico decimali e stamparle. Poi il programma torna a 10 e chiede nuovi dati. Si noti che alla linea 10 vengono chiesti 3 dati di input uno dopo l'altro con una sola parola chiave INPUT; le richieste sono separate tra loro dalla virgola. Alla linea 10 sono separate con la virgola anche le frasi esplicative tra virgolette e le variabili di input (cosa che in alcune implementazioni del Basic non è consentita). Dato che la virgola fa saltare a metà linea, in questo caso si ottiene una richiesta di variabile per riga. Provate a sostituire alla 10 le virgole con i punti e virgola e provate nuovamente il programma; provate ad operare la sostituzione in tutti i modi possibili e riflettete sui risultati ottenuti.

```
   1 REM misure
   5 LET cyd=91.4399: LET cft=30
.48: LET cin=2.54
   6 PRINT "1 iarda=cm ";cyd,"1
piede=cm ";cft,"1 pollice=cm ";c
in
  10 INPUT " iarde? ",yd," piedi
? ",ft," pollici? ",in
  40 GO SUB 2000: REM stampa i v
alori
  50 PRINT " = ";
  70 GO SUB 1000: REM sistemazio
ne dati
  80 GO SUB 2000: REM stampa i v
alori sistemati
  90 PRINT
  95 GO SUB 3000
 100 GO TO 10
1000 REM sistemazione yd, ft, in
 nella loro forma normale
1010 LET in=36*yd+12*ft+in: REM
trasformazione in pollici
1020 GO SUB 4000
1030 LET s=SGN in: LET in=ABS in
: REM s contiene segno di in, in
 diventa positivo
1050 LET ft=INT (in/12): LET in=
(in-12*ft)*s: REM ora pollici va
nno bene
1060 LET yd=INT (ft/3)*s: LET ft
=ft*s-3*yd: RETURN
2000 REM stampa yd,ft,in
2010 PRINT yd;" iarde ";ft;" pie
di ";in;" pollici";: RETURN
3000 REM stampa sistema metrico
3005 PRINT "corrispondono a:"
3010 PRINT " metri ";m;" centime
tri ";cm;" millimetri ";mm
3020 RETURN
4000 REM trasformazione in metri
co decimale
```

```
4010 LET cm=in*2.54: LET s=SGN c
m: LET cm=ABS cm
4020 LET m=INT (cm/100): LET cm=
cm-m*100: LET m=m*s
4025 LET mm=cm-10*INT (cm/10): L
ET mm=s*mm
4030 LET cm=s*INT cm: RETURN
```

Questo programma non finisce mai, cioè non viene analizzata una particolare risposta alla richiesta di dati per decidere se fermare il programma. Per uscire dal programma potete rispondere alla richiesta di input premendo STOP. Dato che il dato richiesto è numerico, con STOP il programma si ferma. Se, invece, la richiesta di input è per una stringa, prima di premere STOP si deve uscire dalle virgolette verso sinistra (usando il tasto freccia), oppure cancellare le virgolette.

I risultati riportati sono stati ottenuti con il comando COPY dopo aver fermato il programma.

```
1 iarda=cm 91.4399
1 piede=cm 30.48
1 pollice=cm 2.54
45 iarde 32 piedi 12 pollici
 = 56 iarde 0 piedi 0 pollici
corrispondono a:
 metri 51 centimetri 20 millimet
ri 0.6399939
3 iarde 78 piedi 345 pollici
 = 38 iarde 1 piedi 9 pollici
corrispondono a:
 metri 35 centimetri 25 millimet
ri 8.0600004
```

PANGOLINO

Si tratta di un gioco. Il programma contiene nelle frasi DATA delle linee da 2000 a 2040 le caratteristiche distintive (tre in tutto) di alcuni animali. Ogni caratteristica è seguita da due numeri che servono come indice per far puntare o a un'altra caratteristica o al nome di un animale. Seguono i nomi di quattro animali. Alla linea 10 viene assegnato alla variabile NQ il valore 100; tale numero rappresenta il numero totale di caratteristiche e di animali che possono essere memorizzati. Alla linea 15 vengono dimensionati: il vettore Q$ che può contenere 100 stringhe di 50 caratteri ciascuna, e la matrice numerica A di 100 righe e 2 colonne. La variabile QF, posta al valore 8 alla linea 20, viene usata per l'algoritmo di caricamento di Q$ e di A. Dopo l'esecuzione delle istruzioni delle linee da 30 a 70, la situazione dei dati in memoria è la seguente:

N	Q$(N)	A(N,1)	A(N,2)
1	vive nel mare	4	2
2	è squamoso	3	5
3	mangia le formiche	6	7
4	una balena	0	0
5	un rettile	0	0
6	un pangolino	0	0
7	una formica	0	0

Come potete notare, mentre alle caratteristiche corrispondono due puntatori alle linee in base al valore dell'indice n, ai nomi degli animali corrispondono puntatori con valore zero.

Il gioco inizia alla linea 100 chiedendo di pensare a un animale. La PAUSE 0 della linea 120 serve per creare un ciclo di attesa fino a quando viene premuto un tasto qualsiasi. Dalla linea 130 alla 380 vengono proposte alcune caratteristiche degli animali e, a seconda della risposta, viene proposta una nuova caratteristica o il nome di un animale. Se l'animale non viene accettato ed è ancora disponibile memoria, viene chiesto il nome di un nuovo animale da inserire e le sue caratteristiche distintive. In questo modo viene arricchito il vocabolario del gioco.

```
   5 REM pangolini
  10 LET nq=100: REM numero dell
e domande e degli animali
  15 DIM q$(nq,50): DIM a(nq,2):
 DIM r$(1)
  20 LET qf=6
  30 FOR n=1 TO qf/2-1
  40 READ q$(n): READ a(n,1): RE
AD a(n,2)
  50 NEXT n
  60 FOR n=n TO qf-1
  70 READ q$(n): NEXT n
 100 REM inizio gioco
 110 PRINT "Pensa a un animale."
,"Premi un tasto per continuare.
"
 120 PAUSE 0
 130 LET c=1: REM prima domanda
 140 IF a(c,1)=0 THEN GO TO 300
 150 LET p$=q$(c): GO SUB 910
 160 PRINT "?": GO SUB 1000
 170 LET in=1: IF r$="s" OR r$="
S" THEN GO TO 210
 190 LET in=2: IF r$="n" THEN GO
 TO 210
 200 IF r$<>"N" THEN GO TO 150
 210 LET c=a(c,in): GO TO 140
 300 REM animali
 310 PRINT "Stai pensando a ";
```

```
 320 LET p$=q$(c): GO SUB 900: P
RINT "?"
 330 GO SUB 1000
 340 IF r$="s" OR r$="S" THEN GO
 TO 400
 360 IF r$="n" OR r$="N" THEN GO
 TO 500
 380 PRINT "Rispondimi a tono qu
ando          ti parlo.": GO TO 30
0
 400 REM supposizione
 410 PRINT "Ho pensato abbastanz
a.": GO TO 800
 500 REM nuovi animali
 510 IF qf>nq-1 THEN PRINT "Sono
 sicuro che il tuo animale  e' m
olto interessante, ma ora   non
ho posto per lui.": GO TO 800
 520 LET q$(qf)=q$(c): REM muovi
 il vecchio animale
 530 PRINT "Che cosa e'? ": INPU
T q$(qf+1)
 540 PRINT "Fammi una domanda ch
e mi permet-ta di distinguere tr
a ";
 550 LET p$=q$(qf): GO SUB 900:
PRINT " e ";
 560 LET p$=q$(qf+1): GO SUB 900
: PRINT ""
 570 INPUT s$: LET i=LEN s$
 580 IF s$(i)="?" THEN LET i=i-1
 590 LET q$(c)=s$( TO i): REM do
manda
 600 PRINT "Quale risposta per "
;
 610 LET p$=q$(qf+1): GO SUB 900
: PRINT "?"
 620 GO SUB 1000
 630 LET in=1: LET io=2: REM dom
anda per vecchio e nuovo animale
 640 IF r$="s" OR r$="S" THEN GO
 TO 700
 660 LET in=2: LET io=1
 670 IF r$="n" OR r$="N" THEN GO
 TO 700
 680 PRINT "Cosi' non va bene.":
 GO TO 600
 700 REM aggiorna risposte
 710 LET a(c,in)=qf+1: LET a(c,i
o)=qf
 720 LET qf=qf+2: REM posto libe
ro per animali
 730 PRINT "Mi hai giocato."
 800 REM ancora
 810 PRINT "Vuoi giocare ancora
(s/n)? ": GO SUB 1000
 820 IF r$="s" OR R$="S" THEN GO
 TO 100
 840 STOP
 900 REM stampa eliminando spazi
 inutili
```

```
 905 PRINT " ";
 910 FOR n=50 TO 1 STEP -1
 920 IF p$(n)<>" " THEN GO TO 94
0
 930 NEXT n
 940 PRINT p$( TO n);: RETURN
1000 REM legge risposta
1010 INPUT r$: IF r$="" THEN RET
URN
1020 LET r$=r$(1): RETURN
2000 REM animali iniziali
2010 DATA "Vive nel mare ",4,2
2020 DATA "E' squamoso ",3,5
2030 DATA "Mangia le formiche ",
5,7
2040 DATA "una balena","un retti
le","un pangolino","una formica"
```

Quando siete stanchi di giocare potete rendervi conto della situazione delle variabili in memoria scrivendo in modo immediato quanto segue:

FOR J=1 TO QF:PRINT Q$(J);A(J,1);A(J,2):NEXT J

se avete giocato molto dovrete premere un tasto per avere lo scrolling del video.

Anche in questo programma si sfrutta la tecnica dei sottoprogrammi. Si consideri la routine da 900 a 940 che serve per stampare stringhe di 50 caratteri eliminando gli spazi inutili, e la routine da 1000 a 1020 per ricevere le risposte ai quesiti posti.

BANDIERA

Questo programma disegna a colori la bandiera del Regno Unito. Alla linea 10 vengono inizializzate 3 variabili per i colori: R=2 per rosso, W=7 per bianco, B=1 per blù. Alla linea 20 viene messo a nero il bordo del video, a blù lo sfondo e a bianco la scrittura. Dalla 30 alla 70, usando il comando INVERSE 1, che fa usare il colore dello sfondo al posto del colore dell'inchiostro e viceversa in tutte le operazioni di scrittura, e ponendo lo sfondo a nero all'interno dei comandi PLOT e DRAW, si ottiene di ridurre le dimensioni del rettangolo blù della bandiera facendo diventare nere le righe del video da 40 a 0 (coordinate riferite ai pixel); alla linea 70 con INVERSE 0 si ritorna alle condizioni normali per sfondo e scrittura. Vale la pena riflettere su questo annerimento, dato che i comandi PLOT e DRAW sono usati solo a scopo di riempimento colore e non di disegno di pixel. Infatti il ciclo FOR della linea 50 procede per N che varia da 40 a 0 con STEP=—8. Questo significa che viene messo il pixel di posizione 7,40 al colore dello sfondo, per effetto del comando INVERSE 1, che è poi il colore di tutti gli altri 63 pixel della posizione di stampa. Senza PAPER 0 all'interno del comando PLOT non si sarebbe visto alcun cambiamento; PAPER 0 fa sì che il colore dello sfondo della posizione di stampa (64 pixel) passi da blù a nero. Questo effetto si prolunga su tutta la riga

grazie al comando DRAW che segue. Notate il diverso significato della coppia di numeri che seguono rispettivamente i comandi PLOT e DRAW; nel primo caso essi rappresentano le coordinate x e y di un pixel, nel secondo sono la distanza relativa in pixel del punto di arrivo della linea dal punto di partenza. Per questa ragione nelle linee seguenti del programma si trovano dopo DRAW anche numeri negativi, cosa che non può avvenire con PLOT.

Dalla linea 100 alla linea 490 vengono disegnate le parti bianche della bandiera; INK è rimasto al valore 7.

Dalla linea 500 alla linea 690 vengono disegnate, con tecnica analoga a quella descritta precedentemente, le parti rosse della bandiera.

```
   5 REM bandiera del regno unit
o
  10 LET r=2: LET w=7: LET b=1
  20 BORDER 0: PAPER b: INK w: C
LS
  30 REM nero in fondo al video
  40 INVERSE 1
  50 FOR n=40 TO 0 STEP -8
  60 PLOT PAPER 0;7,n: DRAW PAPE
R 0;241,0
  70 NEXT n: INVERSE 0
 100 REM disegna nelle parti bia
nche
 105 REM San Giorgio
 110 FOR n=0 TO 7
 120 PLOT 104+n,175: DRAW 0,-35
 130 PLOT 151-n,175: DRAW 0,-35
 140 PLOT 151-n,48: DRAW 0,35
 150 PLOT 104+n,48: DRAW 0,35
 160 NEXT n
 200 FOR n=0 TO 11
 210 PLOT 0,139-n: DRAW 111,0
 220 PLOT 255,139-n: DRAW -111,0
 230 PLOT 255,84+n: DRAW -111,0
 240 PLOT 0,84+n: DRAW 111,0
 250 NEXT n
 300 REM San Andrea
 310 FOR n=0 TO 35
 320 PLOT 1+2*n,175-n: DRAW 32,0
 330 PLOT 224-2*n,175-n: DRAW 16
,0
 340 PLOT 254-2*n,48+n: DRAW -32
,0
 350 PLOT 17+2*n,48+n: DRAW 16,0
 360 NEXT n
 370 FOR n=0 TO 19
 380 PLOT 185+2*n,140+n: DRAW 32
,0
 390 PLOT 200+2*n,83-n: DRAW 16,
0
 400 PLOT 39-2*n,83-n: DRAW 32,0
 410 PLOT 54-2*n,140+n: DRAW -16
,0
 420 NEXT n
```

```
 425 REM aggiunge bit extra
 430 FOR n=0 TO 15
 440 PLOT 255,160+n: DRAW 2*n-30,0
 450 PLOT 0,63-n: DRAW 31-2*n,0
 460 NEXT n
 470 FOR n=0 TO 7
 480 PLOT 0,160+n: DRAW 14-2*n,0
 485 PLOT 255,63-n: DRAW 2*n-15,0
 490 NEXT n
 500 REM strisce rosse
 510 INVERSE 1
 520 REM San Giorgio
 530 FOR n=96 TO 120 STEP 8
 540 PLOT PAPER r;7,n: DRAW PAPER r;241,0
 550 NEXT n
 560 FOR n=112 TO 136 STEP 8
 570 PLOT PAPER r;n,168: DRAW PAPER r;0,-113
 580 NEXT n
 600 REM San Patrizio
 610 PLOT PAPER r;170,140: DRAW PAPER r;70,35
 620 PLOT PAPER r;179,140: DRAW PAPER r;70,35
 630 PLOT PAPER r;199,83: DRAW PAPER r;56,-28
 640 PLOT PAPER r;184,83: DRAW PAPER r;70,-35
 650 PLOT PAPER r;86,83: DRAW PAPER r;-70,-35
 660 PLOT PAPER r;72,83: DRAW PAPER r;-70,-35
 670 PLOT PAPER r;56,140: DRAW PAPER r;-56,28
 680 PLOT PAPER r;71,140: DRAW PAPER r;-70,35
 690 INVERSE 0: PAPER 0: INK 7
```

Riportiamo un più semplice programma per ottenere la bandiera italiana. Provate voi a scrivere i programmi per ottenere le bandiere degli altri stati.

```
   5 REM bandiera italiana
  10 LET r=2: LET v=4: LET b=7
  15 LET s=5
  20 BORDER s  PAPER b: INK s: CLS
  30 REM rettangolo bandiera bianco
  40 INVERSE 1
  50 FOR n=40 TO 0 STEP -8
  60 PLOT PAPER s;7,n: DRAW PAPER s;241,0
  70 NEXT n: INVERSE 0
```

```
 100 REM disegna parte verde
 105 INVERSE 1
 110 FOR n=0 TO 15
 120 PLOT PAPER v;0,175-n*8: DRA
W PAPER v;75,0
 160 NEXT n
 170 INVERSE 0
 200 REM disegna parte rossa
 205 INVERSE 1
 210 FOR n=0 TO 15
 220 PLOT PAPER r;180,175-n*8: D
RAW PAPER r;75,0
 230 NEXT n
 240 INVERSE 0
 250 FOR k=1 TO 47
 260 READ x,y: LET x=x/2: BEEP x
,y
 270 NEXT k
 280 PAUSE 50: RESTORE : GO TO 2
50
 300 STOP
1000 DATA 1,2,0.75,2,0.25,4,2,2
1010 DATA 1,11,0.75,11,0.25,12,2
,11
1020 DATA 1,11,0.75,14,0.25,12,2
,11
1030 DATA 1,9,0.75,11,0.25,9,2,7
1040 DATA 1,11,0.75,11,0.25,11,2
,6
1050 DATA 0.75,7,0.25,9,0.75,7,0
.25,6
1060 DATA 2,4,1,7,0.75,6,0.25,7
1070 DATA 2,9,1,2,3,11
1080 DATA 1,2,0.75,2,0.25,4,2,2
1090 DATA 1,11,0.75,11,0.25,12,2
,11
1100 DATA 1,11,0.75,14,0.25,12,2
,11
1110 DATA 0.5,11,0.5,9,0.5,11,0.
5,7
```

Provate il programma, sentirete anche un pò di musica.

Per interrompere premete BREAK.

IMPICCATO

Si tratta di un gioco all'inizio uno dei giocatori deve introdurre una parola senza farla vedere agli altri, dopo viene mostrata sul video una linea di trattini, uno per ogni lettera della parola, e gli altri giocatori devono proporre una dopo l'altra le lettere che formano la parola. All'inizio compare sulla parte destra del video un omino, mentre il gioco procede, ad ogni lettera proposta e che non fa parte della parola da indovinare, procede la costruzione di una forca di fianco all'omino. Se gli errori sono troppi (ne bastano 8) l'omino finisce impiccato, mentre se la parola viene indovinata l'omino salta, tutto contento, oltre la forca.

Il programma è interessante perchè mostra come si produce il movimento sul video e come si può programmare un disegno memorizzando i parametri per PLOT e DRAW con frasi DATA.

Dalla linea 2000 alla linea 2030 sono memorizzate 8 quaterne di numeri che sono ordinatamente le coordinate X0 e Y0 di PLOT e i parametri X e Y di DRAW necessari per costruire in 8 passi la forca. La parte di programma che utilizza questi DATA si trova dalla linea 420 alla linea 450. Questo è un modo molto semplice e sintetico per ottenere un disegno.
L'omino viene disegnato dal sottoprogramma che sta dalla linea 1000 alla linea 1100. Notate che viene disegnato senza bocca; questa viene aggiunta con espressione diversa a seconda degli avvenimenti.

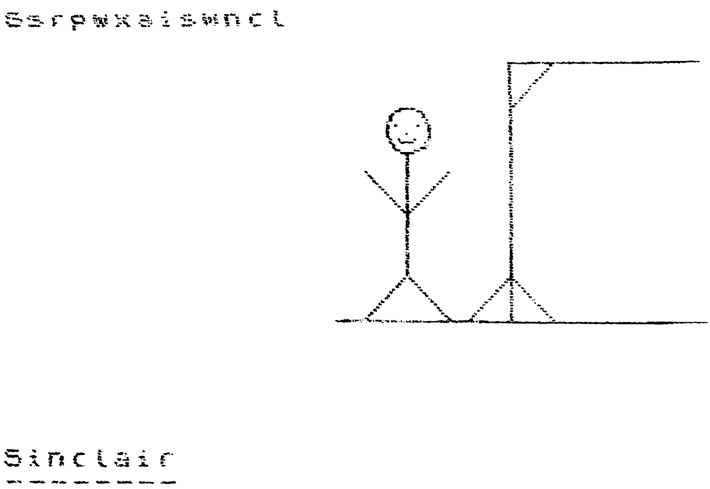
All'inizio viene chiesta la parola da indovinare (linea 25). Poi viene disegnato a destra l'omino, dando il valore 240 alla coordinata di riferimento X e usando il sottoprogramma 1000. Dalla 160 alla 180 vengono scritti tanti trattini quante sono le lettere della parola da indovinare; i trattini vengono poi sostituiti dalle lettere indovinate.
Dalla linea 200 alla 300 si ha il colloquio per avere una lettera, il suo controllo, il salto alla linea 500 se la parola è stata completamente indovinata, la costruzione di un pezzo di forca se la lettera non è stata indovinata. Alla linea 410 si controlla il numero dei tentativi errati, e, se sono 9, dato che la forca è completamente disegnata, l'omino viene impiccato, alle linee da 600 a 730.
L'impiccagione comporta un movimento delle braccia, delle gambe e della bocca del povero omino. Questo movimento è ottenuto con la tecnica dell'overprinting. Prima viene cancellata la vecchia posizione e poi viene disegnata la nuova. Il comando OVER 1 combina il carattere presente in una posizione di stampa con il nuovo carattere che si stampa in questo modo: la matrice di punti (64 pixel) del vecchio carattere viene confrontata con la matrice di punti del nuovo carattere; dove si trovano due bit uguali (tutti e due 0 o tutti e due 1) si ha 0 e dove si trovano due bit diversi si ha 1. Usando un carattere nuovo uguale al vecchio si ottiene così la cancellazione del vecchio. Il comando OVER 0 annulla l'effetto di OVER 1.

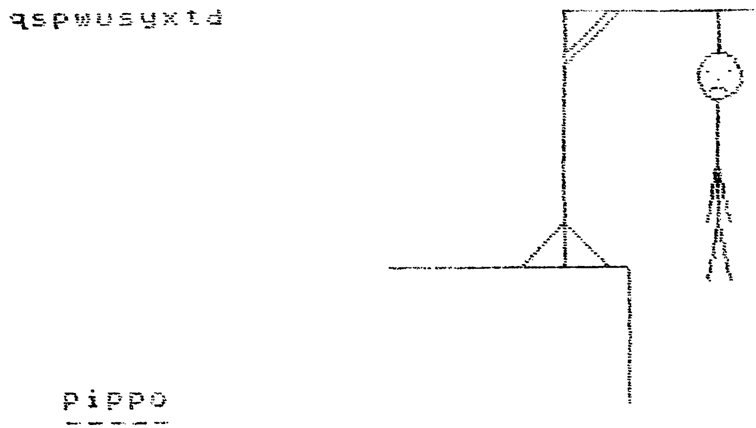
```
   5 REM L'impiccato
  10 REM preparazione video
  20 INK 0: PAPER 7: CLS
  25 INPUT "Scrivi una parola ";
w$: REM parola da indovinare
  30 LET x=240: GO SUB 1000: REM
 disegna un uomo
  40 PLOT 238,128: DRAW 4,0: REM
 bocca
 120 LET i=LEN w$: LET v$=" "
 130 FOR n=2 TO i: LET v$=v$+" "
 140 NEXT n: REM v$=parola indov
inata
 150 LET c=0: LET d=0: REM conta
tore
 160 FOR n=0 TO i-1
 170 PRINT AT 20,n;"-";
```

```
180 NEXT n: REM scrive - al pos
to delle lettere
200 INPUT "Indovina una lettera
: ";g$
210 IF g$="" THEN GO TO 200
220 LET g$=g$(1): REM solo prim
a lettera
230 PRINT AT 0,c;g$
240 LET c=c+1: LET u$=v$
250 FOR n=1 TO i: REM aggiorna
parole indovinate
260 IF w$(n)=g$ THEN LET v$(n)=
g$
270 NEXT n
280 PRINT AT 19,0;v$
290 IF v$=w$ THEN GO TO 500: RE
M parole indovinate
300 IF v$<>u$ THEN GO TO 200: R
EM risposta esatta
400 REM continua a disegnare la
forca
410 IF d=8 THEN GO TO 600: REM
impiccato
420 LET d=d+1
430 READ x0,y0,x,y
440 PLOT x0,y0: DRAW x,y
450 GO TO 200
500 REM uomo libero
510 OVER 1: REM cancella uomo
520 LET x=240: GO SUB 1000
530 PLOT 238,128: DRAW 4,0: REM
bocca
540 OVER 0: REM ridisegna uomo
550 LET x=146: GO SUB 1000
560 PLOT 143,129: DRAW 6,0,PI/2
: REM sorride
570 GO TO 800
600 REM boia
610 OVER 1: REM cancella pavime
nto
620 PLOT 255,65: DRAW -48,0
630 DRAW 0,-48: REM toglie pavi
mento
640 PLOT 238,128: DRAW 4,0: REM
cancella bocca
650 REM muove gli arti
655 REM braccia
660 PLOT 255,117: DRAW -15,-15
DRAW -15,15
670 OVER 0
680 PLOT 236,81: DRAW 4,21: DRA
W 4,-21
690 OVER 1: REM gambe
700 PLOT 255,66: DRAW -15,15: D
RAW -15,-15
710 OVER 0
720 PLOT 236,60: DRAW 4,21: DRA
W 4,-21
730 PLOT 237,127: DRAW 6,0,-PI/
2: REM cipiglio
```

```
 740 PRINT AT 19,0;w$
 800 INPUT "Ancora? ";a$
 810 IF a$="" THEN GO TO 850
 820 LET a$=a$(1)
 830 IF a$="n" THEN STOP
 840 IF a$(1)="N" THEN STOP
 850 RESTORE : GO TO 5
1000 REM disegna uomo alla colon
na x
1010 REM testa
1020 CIRCLE x,132,8
1030 PLOT x+4,134: PLOT x-4,134:
 PLOT x,131
1040 REM corpo
1050 PLOT x,123: DRAW 0,-20
1055 PLOT x,101: DRAW 0,-19
1060 REM gambe
1070 PLOT x-15,66: DRAW 15,15: D
RAW 15,-15
1080 REM braccia
1090 PLOT x-15,117: DRAW 15,-15:
 DRAW 15,15
1100 RETURN
2000 DATA 120,65,135,0,184,65,0,
91
2010 DATA 168,65,16,16,184,81,16
,-16
2020 DATA 184,156,68,0,184,140,1
6,16
2030 DATA 204,156,-20,-20,240,15
6,0,-16
```

Seguono due disegni ottenuti con il comando COPY dopo la conclusione felice e infelice del gioco.





CALEIDO

Questo programma disegna un caleidoscopio sul video modificando continuamente disegno e colori.

In realtà appare un disegno che non è disegnato. Alla linea 20 viene assegnato alla variabile Z il valore 22928. Tale numero è l'indirizzo del byte che contiene gli attributi della posizione del video situata alla tredicesima riga (dall'alto) e al diciassettesimo carattere (da sinistra). I 768 byte dedicati agli attributi delle posizioni di stampa sul video partono dall'indirizzo 22528 (22928—22528+1=401; 401 diviso 32 dà 12 con resto 17). Alla linea 20 viene anche posta la variabile L a zero. Dalla linea 30 alla 170 si ha un ciclo percorso 12 volte in base a I. In tale ciclo è contenuto un ciclo interno controllato dalla variabile J che varia da 0 a L; L che parte da 0 viene incrementata di 1 per ogni valore assunto da I. La linea 50 incrementa x (che parte da 0) di 0.03 e calcola per K un valore che risulta compreso tra —127 e +127 (usa infatti il moltiplicatore SIN x). Tale valore di K viene posto come attributo di 8 posizioni di stampa che si spostano con simmetria intorno alle posizioni centrali del video. La variazione di attributo provoca l'effetto del disegno colorato che si modifica continuamente. Il punto Z, origine delle variazioni di attributi si sposta ad ogni ciclo I sulla riga sottostante (Z=Z+32). Terminato un ciclo I, la linea 180 rimanda alla 20 e tutto ricomincia. Per interrompere il programma dovete usare il tasto BREAK.

```
  10 REM caleidoscopio
  15 LET x=0
  20 LET z=22928: LET l=0
  30 FOR i=1 TO 12
  40 FOR j=0 TO l
  50 LET x=x+.03: LET k=INT (127
*SIN x)
  60 POKE z+j,k
  70 POKE z-j-1,k
  80 POKE z+32-31*i+32*j,k
```

```
 90 POKE z-31*i-32*j,k
100 POKE z-33*i+31+32*j,k
110 POKE z-33*i-1-32*j,k
120 POKE z-64*i+31-j,k
130 POKE z-64*i+32+j,k
140 NEXT j
150 LET l=l+1
160 LET z=z+32
170 NEXT i
180 GO TO 20
```

CALENDARIO

Questo programma serve per produrre il calendario di un singolo mese o di un anno dal 1980 al 2000. Il calendario può anche essere ottenuto in stampa sulla ZX PRINTER.

Viene ancora sfruttata la tecnica dei DATA per memorizzare all'interno del programma la descrizione dei mesi, dei giorni della settimana, e il numero di giorni di ogni mese. Se il mese comincia con sabato o domenica, nella stampa del calendario la prima colonna è quella dei sabati o delle domeniche, se il mese inizia con qualunque altro giorno della settimana, la prima colonna è quella dei lunedì. Se si sbagliano le risposte si ha un BEEP sonoro. Notate l'uso del PAUSE 0 per ottenere una pausa che viene interrotta quando si preme un qualunque tasto. L'algoritmo usato è il solito che si basa sul numero intero di settimane e il controllo dei resti quando sia noto che giorno della settimana è il primo giorno del gennaio 1980.

```
  10 PRINT TAB 12;"CALENDARIO"
  20 PRINT ''"Di un mese o di un
 anno (m/a)? "
  30 INPUT z$: IF z$<>"a" AND z$
<>"m" THEN BEEP 1,12: GO TO 30
  40 PRINT ''"Vuoi la copia su s
tampante (s/n)? "
  50 INPUT c$: IF c$<>"s" AND c$
<>"n" THEN BEEP 1,12: GO TO 50
  60 DATA "GENNAIO","FEBBRAIO","
MARZO","APRILE","MAGGIO","GIUGNO
"
  61 DATA "LUGLIO","AGOSTO","SET
TEMBRE","OTTOBRE","NOVEMBRE","DI
CEMBRE"
  62 DATA "Sab","Dom","Lun","Mar
","Mer","Gio","Ven","Sab","Dom"
  63 DATA 31,28,31,30,31,30,31,3
1,30,31,30,31
  70 DIM m$(12,9): DIM d$(9,3):
DIM d(12)
  80 FOR i=1 TO 12: READ m$(i):
NEXT i
  85 FOR i=1 TO 9: READ d$(i): N
EXT i
  90 FOR i=1 TO 12: READ d(i): N
```

```
EXT i
  95 CLS
 100 IF z$="a" THEN GO TO 1000
 200 PRINT "Calendario per un me
se"''"Scrivi mese e anno"''
 210 INPUT "Mese (da 1 a 12) = "
;n: LET n=INT n
 220 IF n<1 OR n>12 THEN BEEP 1,
12: GO TO 210
 230 LET n$=m$(n): PRINT n$
 240 INPUT "Anno (da 1980 a 2000
) = ";y
 250 IF y<1980 OR y>2000 THEN BE
EP 1,12: GO TO 240
 260 PRINT y
 270 PAUSE 50
 280 GO SUB 300
 290 STOP
 300 LET j=0
 310 IF n=1 THEN GO TO 400
 320 LET d(2)=28+(y=4*INT (y/4)
AND y<>2000)
 330 FOR l=1 TO n-1
 340 LET j=j+d(l)
 350 NEXT l
 400 LET j=j+4+(y-1980)*365+INT
((y-1981)/4)
 410 LET j=j+1-7*INT (j/7)
 500 CLS
 510 PRINT TAB 9;m$(n);TAB 19;y
 520 LET b=-2+4*(j-2+(3-j)*(j<3)
)
 530 LET a=5
 540 FOR l=1 TO d(n)
 550 PRINT AT a,b;l
 560 LET b=b+4
 570 IF b>26 THEN LET b=2: LET a
=a+4
 580 NEXT l
 590 LET j=j-(j-3)*(j>3)
 600 LET a=2
 610 FOR l=j TO j+6
 620 PRINT AT 1,a;d$(l)
 630 LET a=a+4
 640 NEXT l
 650 FOR a=15 TO 239 STEP 32
 660 PLOT a,0
 670 DRAW 0,167
 680 NEXT a
 690 FOR a=0 TO 167 STEP 32
 700 PLOT 15,a
 710 DRAW 224,0
 720 NEXT a
 730 IF c$="s" THEN COPY
 740 RETURN
1000 PRINT TAB 6;"Calendario per
 un anno",,,
1010 INPUT "Anno (da 1980 a 2000
) = ";y
```

```
1020 IF y<1900 OR y>2000 THEN BE
EP 1,12: GO TO 1010
1030 PRINT y
1035 IF c$="s" THEN LPRINT TAB 7
;"Calendario del ";y: LPRINT : L
PRINT : GO TO 1060
1037 PRINT '"Per cambiare mese p
remi un tasto"''"Premi un tasto
per continuare"
1040 PAUSE 0
1060 FOR n=1 TO 12
1070 GO SUB 300
1075 IF c$="s" THEN LPRINT : LPR
INT : GO TO 1090
1078 PAUSE 0
1090 NEXT n
```

Segue un esempio di stampa del calendario del mese di marzo dell'anno 1983.

MARZO 1983

Lun	Mar	Mer	Gio	Ven	Sab	Dom
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

DISEGNI

Questo programma traccia dei disegni che ruotano sul video. All'inizio vengono chiesti:

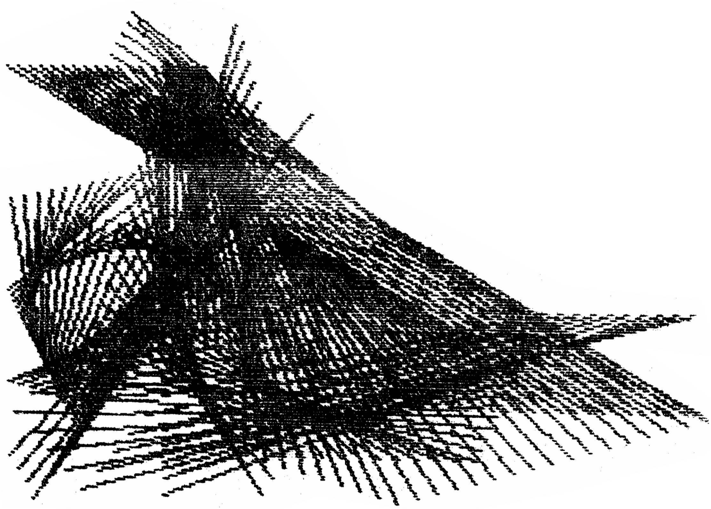
S colore dello sfondo,usato anche per il bordo
D colore del disegno
V1 primo parametro velocità movimento punti
V2 secondo parametro velocità movimento punti
P1 primo parametro intervallo per cambio velocità
P2 secondo parametro intervallo per cambio velocità.

Alla linea 6 si ha il messaggio che dice di premere un tasto per proseguire quando è terminato un disegno. Infatti alla 100 si ha un PAUSE 0, alla fine di un disegno; se si preme un tasto viene pulito il video e il programma ricomincia dalla linea 10 con gli stessi parametri.

La tecnica usata è la seguente:

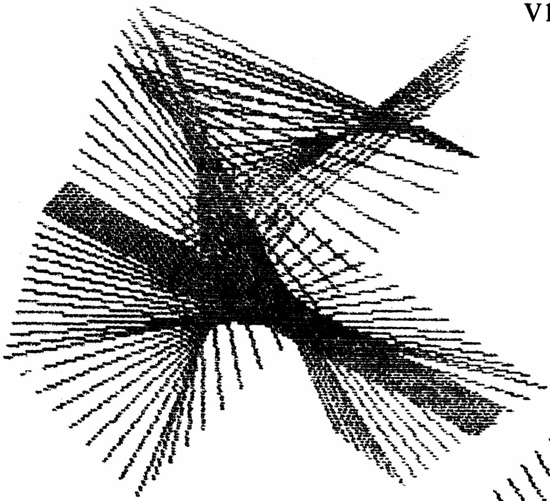
— alla 8 vengono predisposti i colori per il video
— alla 10 vengono calcolate in modo random le coordinate di due punti, in pixel
— alla 15 viene definita una funzione che genera numeri a caso minori dell'argomento passato
— alla 20 viene richiamato il sottoprogramma in 500 che genera 4 numeri a caso usando la funzione definita e i parametri V1 e V2 e genera un numero a caso per N usando i parametri temporali P1 e P2
— da 25 a 90 di ha il ciclo di disegno della linea tra i due punti; in questo ciclo quando N diventa zero viene richiamato il sottoprogramma e viene modificata la velocità ed N stesso
— per terminare viene generato un numero a caso minore di 199 (linea 80) e quando questo numero risulta 1 il disegno termina.

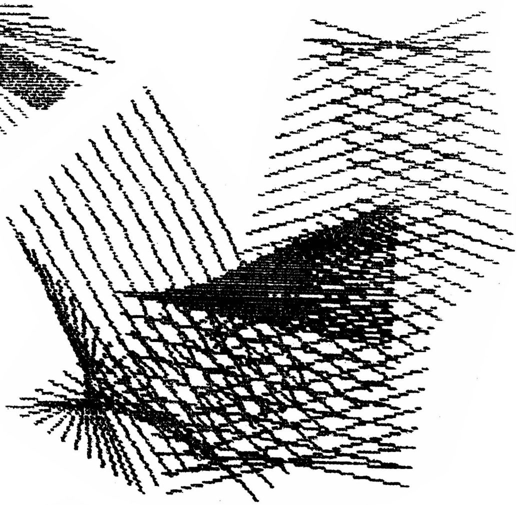
Le variabili in gioco sono quindi molte. Potete provare a variare P1, P2, V1 e V2 e vedrete comparire i più svariati disegni. Quando il disegno termina potete premere BREAK e ottenere con COPY il disegno sulla stampante come si è fatto per gli esempi che seguono.



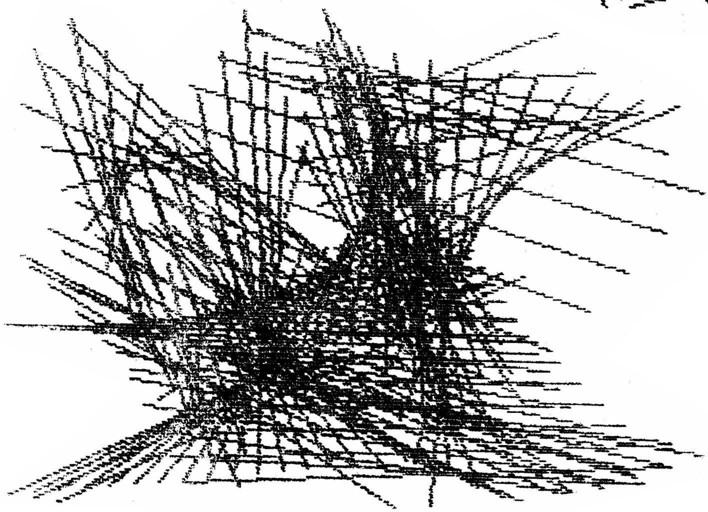
V1 = 15, V2 = 7, P1 = 20, P2 = 10

V1 = 15, V2 = 7, P1 = 10, P2 = 5





V1 = 15, V2 = 7, P1 = 40, P2 = 20



V1 = 30, V2 = 14, P1 = 20, P2 = 10

Listato programma disegni

```
   1 PRINT "DISEGNI ROTANTI"
   2 INPUT "Colore sfondo (da 1
a 7)? ";s: PRINT '"Sfondo ";s
   3 INPUT "Colore disegno (da 1
a 7)? ";d: PRINT '"Disegno ";d
   4 PRINT '"Parametri velocita'
movimento punti:": INPUT "v1=
";v1: PRINT "v1=";v1;" ";: INPUT
"v2=";v2: PRINT "v2=";v2
   5 PRINT '"Parametri intervall
o per cambio velocita':": INPUT
"p1=";p1: PRINT "p1=";p1;" ";: I
NPUT "p2=";p2: PRINT "p2=";p2
   6 PRINT '"Quando il disegno e
' terminato premere un tasto pe
r continuare."
   8 PAUSE 100: BORDER s: PAPER
s: CLS : INK d
  10 LET x=INT (RND*256): LET y=
INT (RND*176): LET w=INT (RND*25
6): LET z=INT (RND*176)
  15 DEF FN f(x)=INT (RND*x)
  20 GO SUB 500
  25 REM ciclo
  30 LET n=n-1
  35 IF n=0 THEN GO SUB 500
  40 PLOT x,y
  45 DRAW w-x,z-y
  50 IF x+f1>255 OR x+f1<0 THEN
LET f1=-f1
  55 IF y+f2>175 OR y+f2<0 THEN
LET f2=-f2
  60 IF w+f3>255 OR w+f3<0 THEN
LET f3=-f3
  65 IF z+f4>175 OR z+f4<0 THEN
LET f4=-f4
  70 LET x=x+f1: LET y=y+f2: LET
 w=w+f3: LET z=z+f4
  75 REM controllo condizione
  80 LET c=FN f(200)
  85 IF c=1 THEN GO TO 100
  90 GO TO 25
 100 PAUSE 0
 110 CLS : GO TO 10
 500 LET f1=FN f(v1)-v2: LET f2=
FN f(v1)-v2: LET f3=FN f(v1)-v2:
 LET f4=FN f(v1)-v2
 510 LET n=FN f(p1)+p2
 520 RETURN
```

RETTANGOLO

Si ha un rettangolo di 6 per 4 caselline in ogni casellina sta un numero. I numeri presenti vanno da 1 a 24 e sono in disordine. Il gioco consiste nel mettere in ordine i numeri partendo con 1 nell'angolo in alto a sinistra.

```
RETTANGOLO MAGICO

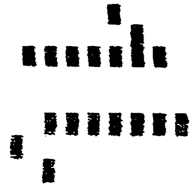
21 23 3  18 15 6
9  7  10 4  1  5
22 14 2  12 13 11
20 19 8  16 17 24
```

Come si vede dall'immagine video sopra riportata, che rappresenta il quadro iniziale di un gioco, nella posizione centrale dei primi 4 numeri in alto a sinistra sta un simbolo speciale, che chiamiamo puntatore. Usando i 4 tasti freccia (5, 6, 7 e 8) si può spostare il puntatore. Le posizioni consentite per il puntatore sono: tutti gli spazi tra le caselline sulle linee orizzontali (prima, terza, quinta e settima) e tutti gli spazi corrispondenti alle colonne pari (seconda, quarta, sesta, ottava e decima) sulle linee orizzontali che separano le caselline dei numeri. Le mosse consentite per mettere in ordine i numeri sono: lo scambio di due numeri su una riga orizzontale e la rotazione in senso orario di 4 numeri. Per muovere il giocatore preme il tasto corrispondente allo zero: se il puntatore si trova tra 2 numeri, questi sono scambiati tra loro, se il puntatore si trova tra 4 numeri, come nell'esempio sopra, si ha la rotazione dei 4 numeri. Durante il movimento dei numeri si ha l'emissione di un BEEP sonoro. All'inizio del gioco compare la scritta "QUADRATO MAGICO" su sfondo nero e poi si ha una breve attesa durante la quale il calcolatore prepara in modo casuale la disposizione dei numeri e poi traccia il rettangolo emettendo un BEEP sonoro per ogni casellina che compare.

```
  10 BORDER 0: PAPER 0: INK 7: B
RIGHT 1: CLS : PRINT AT 0,6; PAP
ER 1;" RETTANGOLO MAGICO "
  11 LET fl=0: GO SUB 400: GO SU
B 150: GO SUB 250
  12 GO SUB 350
  13 IF fl=2 THEN INPUT "Premi E
NTER per giocare ancora "; LINE
z$: RUN
  14 GO TO 12
 100 REM quadro
 101 DIM b$(2)
 102 LET b$=STR$ b(y,x)
 103 PRINT AT 2+2*y,3+3*x; PAPER
6; INK 0;b$
 104 BEEP .2,6*y+x: RETURN
 150 REM preparazione
 151 DIM b(4,6)
 152 FOR n=1 TO 24
 153 LET x=1+INT (RND*6): LET y=
1+INT (RND*4)
 154 IF b(y,x) THEN GO TO 153
 155 LET b(y,x)=n: NEXT n
```

```
 156 LET cnt=0: LET x3=1: LET y3
=1
 157 RETURN
 200 REM rotazione
 201 LET b=b(y3,x3): LET b(y3,x3
)=b(y3+1,x3)
 202 LET b(y3+1,x3)=b(y3+1,x3+1)
 203 LET b(y3+1,x3+1)=b(y3,x3+1)
 204 LET b(y3,x3+1)=b
 205 FOR y=y3 TO y3+1: FOR x=x3
TO x3+1: GO SUB 100: NEXT x: NEX
T y
 206 LET cnt=cnt+1: PRINT AT 0,0
;cnt
 207 RETURN
 250 FOR y=1 TO 4: FOR x=1 TO 6
 251 GO SUB 100
 252 NEXT x: NEXT y: PRINT AT 5,
8;CHR$ 144
 253 RETURN
 300 REM scambio
 301 LET b=b(y3+.5,x3): LET b(y3
+.5,x3)=b(y3+.5,x3+1)
 302 LET b(y3+.5,x3+1)=b
 303 FOR x=x3 TO x3+1: LET y=y3+
 5
 304 GO SUB 100: NEXT x
 305 LET cnt=cnt+1: PRINT AT 0,0
 cnt
 308 RETURN
 350 REM spostamento
 351 PAUSE 10: IF INKEY$<>"" THE
N GO TO 351
 352 LET z$=INKEY$
 353 IF z$="" THEN GO TO 352
 354 BEEP .2,30: PRINT AT 3+2*y3
,5+3*x3;" "
 355 IF z$="8" THEN LET x3=x3+1:
 IF x3>5 THEN LET x3=5: GO TO 36
0
 356 IF z$="5" THEN LET x3=x3-1:
 IF x3<1 THEN LET x3=1: GO TO 36
0
 358 IF z$="6" THEN LET y3=y3+0.
5: IF y3>3.5 THEN LET y3=3.5: GO
 TO 360
 359 IF z$="7" THEN GO TO 500
 360 PRINT AT 3+2*y3,5+3*x3;CHR$
 144
 361 IF z$="1" THEN LET fl=2: RE
TURN
 365 IF z$="0" AND ABS (y3-INT (
y3))=0 THEN GO SUB 200: RETURN
 370 IF z$="0" AND ABS (y3-INT (
y3))<>0 THEN GO SUB 300: RETURN
 380 RETURN
 400 DATA 8,4,254,0,0,127,32,16
 401 FOR n=0 TO 7: READ a: POKE
USR "a"+n,a: NEXT n
 402 RETURN
 500 IF y3>=1 THEN LET y3=y3-0.5
 501 GO TO 360
```

Tra i punti interessanti del programma si può citare la creazione di un carattere grafico personale per il puntatore. Questo carattere viene costruito usando gli 8 numeri che compaiono nel DATA della linea 400 e che rappresentano i contenuti decimali degli 8 byte che disegnano il carattere per punti (pixel), come appare dallo schema riportato.



```
00001000
00000100
11111110
00000000
00000000
01111111
10000000
01000000
```

Il puntatore viene costruito in modo da corrispondere al tasto "A" che corrisponde a CHR$ 144, alla linea 401 con il ciclo:

READ A:POKE USR "A"+N,A

e viene richiamato usando PRINT CHR$ 144.

Il programma si vale della tecnica dei sottoprogrammi: alla linea 11 viene predisposto il FLAG FL al valore zero (se al valore 2 serve per terminare il gioco) e vengono usati i sottoprogrammi a:

400 per costruire il puntatore,

150 per preparare in modo casuale i contenuti delle caselline dei numeri nella matrice B(Y,X),

250 per disegnare il quadro servendosi anche del sottoprogramma a 100.

Dalla linea 12 alla 14 si svolge il gioco e viene usato il sottoprogramma a 350 che serve per leggere il movimento del giocatore e quindi:

- spostare il cursore,
- eseguire il movimento dei numeri,
- far ricominciare il gioco.

Il giocatore può:

- spostare il cursore usando i tasti freccia,
- eseguire uno scambio o una rotazione di numeri usando il tasto zero,
- ricominciare il gioco usando il tasto uno.

FILE DI DATI

Segue un esempio di come si può preparare ed usare un file di dati, cioè un file che si memorizza e si rilegge usando rispettivamente:

SAVE "nome" DATA "nome array"()

LOAD "nome" DATA "nome array"().

Nome è il nome usato per memorizzare su nastro, mentre nome array è il nome di un array o stringa o numerico usato per il file in memoria.

Il programma che segue e che chiamiamo: "CARICAFILE" serve per preparare un file di dati avente le seguenti caratteristiche:

- può contenere fino a 1000 caratteri,
- ha come primo carattere un "?",
- ogni record inizia con "?", i campi all'interno del record sono di lunghezza variabile e sono separati tra loro da un "!",
- il file termina con 2 "?",
- il programma avvisa quando sono rimasti disponibili meno di 80 caratteri per un record,
- per uscire dal programma e memorizzare il file si deve rispondere con "*" alla richiesta del primo dato,
- il record è composto dai campi:Cognome, Nome, Indirizzo, Città, Cap e Telefono.

```
   1 PRINT '"Preparazione file d
i dati."''"Viene dimensionata un
a stringa  f$ a 1000 caratteri.
Si caricanodegli indirizzi. Il p
rogramma   avvisa quando sono di
sponibili  meno di 80 caratteri.
"
   2 PRINT "Ogni record inizia c
on un punto interrogativo ottenu
to con         CHR$ 0. I campi di l
unghezza va-riabile sono separat
i dal carat-tere ! (CHR$33)."
   3 PRINT "Il file inizia con u
n punto in- terrogativo e termin
a con due    punti interrogativi.
"
   4 PRINT "Per terminare il fil
e rispondereal primo dato con *.
 Il file       viene memorizzato su
 nastro con  il nome ""file"", c
ome stringa   f$."
   5 GO SUB 500: GO SUB 600
   6 CLS : PRINT "I campi sono:"
'c$'n$'i$'l$'d$'t$
  10 LET f$(1)=r$: LET k=2
  20 INPUT (c$);a$: IF a$="*" TH
EN GO TO 700
  30 LET p$="": LET p=0: LET a$=
r$+a$: GO SUB 650
  40 INPUT (n$);a$: GO SUB 650
  50 INPUT (i$);a$: GO SUB 650
  60 INPUT (l$);a$: GO SUB 650
  70 INPUT (d$);a$: GO SUB 650
  80 INPUT (t$);a$: GO SUB 650
```

```
  85 IF m-(k+p+2)<0 THEN PRINT "
Il record non entra in f$.": GO
TO 700
  90 LET f$(k TO k+p-1)=p$: LET
k=k+p
  95 IF m-(k+2)<80 THEN PRINT "R
estano liberi ";m-(k+2);" caratt
eri."
 100 GO TO 20
 500 PRINT "Premi un tasto per c
ontinuare.": PAUSE 0: RETURN
 600 LET c$="Cognome ": LET n$="
Nome ": LET i$="Indirizzo ": LET
 l$="Citta' ": LET d$="CAP ": LE
T t$="Tel. "
 610 LET m=1000: LET r$=CHR$ 0:
LET s$=CHR$ 33: DIM f$(m): RETUR
N
 650 LET n=LEN a$: LET p$=p$+a$+
s$: LET p=p+n+1: RETURN
 700 CLS : PRINT "Il file viene
memorizzato."
 710 LET f$(k TO k+1)=r$+r$
 720 SAVE "file" DATA f$()
 730 PRINT "Ferma il nastro"
 740 STOP
```

Nel programma sono inseriti i messaggi per il controllo del nastro. Il file generato deve essere conservato su nastro e usato con il programma LEGGIFILE che segue. Si tratta solo di un esempio, voi potrete fare le applicazioni che desiderate. Alla linea 610 si è posta la dimensione della stringa F$, che serve a contenere il file, a 1000 caratteri, basta cambiare il valore di M per ottenere un file più grande. Alla linea 95 si imposta il controllo dei caratteri ancora disponibili sul numero 80, che può essere modificato secondo le proprie necessità. Segue il contenuto di' F$, ottenuto con un comando immediato di PRINT F$ dopo aver terminato il programma, usato per caricare 3 indirizzi.

```
??Primo!Cliente!Viale Roma 876!M
ilano!21345!02/7865432!?Secondo!
Cliente!Piazza Napoli 90!Varese!
54321!90876543!?Terzo!Fornitore!
Corso Como 56!Milano!21345!87654
321!??
```

Segue il listato del programma "LEGGIFILE" che serve per leggere il file "FILE" nella stringa F$ e stampare i 3 indirizzi.

```
  10 REM lettura e stampa file
  20 PRINT "Monta cassetta con f
ile dati."
  30 PRINT "Avvia il nastro e pr
emi un tastoper continuare.": PA
USE 0
  40 LOAD "file" DATA f$()
```

```
  45 PRINT '"Ferma il nastro."'"
Premi un tasto per continuare.":
 PAUSE 0: CLS
  50 LET k=3: LET r$=CHR$ 0: LET
 s$=CHR$ 33
  55 LET p$=""
  60 IF f$(k)<>s$ THEN LET p$=p$
+f$(k): LET k=k+1: GO TO 60
  70 PRINT p$
  80 LET k=k+1: IF f$(k)=r$ THEN
 GO TO 200
  90 GO TO 55
 200 LET k=k+1: IF f$(k)=r$ THEN
 GO TO 300
 210 PRINT : GO TO 55
 300 PRINT ''"Finito il file"
 310 STOP
```

Il programma è molto semplice; come potete vedere non è necessario dimensionare la stringa F$ usata per leggere il file. Questa stringa viene poi gestita con lo SLICING per reperire i record e i campi controllando i caratteri separatori "?" e "!". Segue il contenuto del video ottenuto con la COPY. Se usate questo programma con un file più lungo, dovrete premere un tasto quando richiesto per ottenere lo SCROLLING del video.

```
Primo
Cliente
Viale Roma 876
Milano
21345
02/7865432

Secondo
Cliente
Piazza Napoli 90
Varese
54321
90876543

Terzo
Fornitore
Corso Como 56
Milano
21345
87654321
```

Con un pò di pratica è possibile fare un uso molto raffinato dei file di dati. Si può gestire la stringa di dati con un programma in linguaggio macchina avendo l'accorgimento di scrivere il programma in modo che quando comincia a girare esegua come prima istruzione il LOAD del file di dati. In questo modo la stringa F$ (dell'esempio) diventa la prima variabile dell'area delle variabili. L'indirizzo dell'inizio dell'area delle variabili si ricava dal puntatore a 2 byte VARS, situato nei byte 23627 e 23628, calcolando: indirizzo=256*PEEK 23628 + PEEK 23627. Tale indirizzo è quello della stringa che contiene il file di dati; aggiungendo il numero dei

byte usati dal sistema come testata della stringa, si punta al primo carattere del file e quindi si può accedere a tutti i caratteri molto velocemente.

Inoltre il programma di caricamento del file può costruire contemporaneamente al file principale di dati anche un secondo file da usare come indice; in questo file indice basta riportare un campo come chiave di ricerca per il record e il valore del puntatore all'inizio del record del file principale (valore noto in fase di caricamento) da usare nello SLICING per accedere a quel record.

Per concludere, con un pò di esercizio, si possono sfruttare tutte le grandi possibilità dello ZX SPECTRUM.

FILE DI BYTE

Segue un esempio di trattamento dei file di byte, cioè di quei file che si memorizzano e si leggono usando i comandi:

SAVE "nome" CODE n,m

LOAD "nome" CODE n,m

dove "nome" è il nome usato per memorizzare su nastro, n è l'indirizzo del primo byte da memorizzare e dove iniziare a rileggere, e m è il numero di byte. Tali comandi prendono un aspetto particolare se n=16384 e m=6912 (6912=6144+768, cioè il numero di byte corrispondenti alla memoria video e alla mappa degli attributi della stessa), e possono essere scritti rispettivamente:

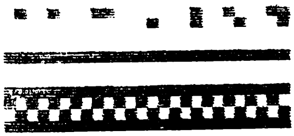
SAVE "nome" SCREEN$ e LOAD "nome" SCREEN$.

Segue il listato del programma "MEMOVIDEO" che crea un semplice disegno colorato e una scritta sul video e memorizza il contenuto del video sul nastro.

```
  10 REM prova memorizzazione vi
deo
  20 CLS
  30 PRINT INK 4;"▀ ▀ ▀▀ ▄ █ ▚ █"
  40 PRINT
  50 PRINT INK 2;"████████████████"
  60 PRINT INK 5;"████████████████"

  70 PRINT INK 4;"▞▞▞▞▞▞▞▞▞▞▞▞▞▞▞▞"
  80 PRINT INK 5;"████████████████"
  90 PRINT : PRINT "PROVA MEMORI
ZZAZIONE VIDEO"
 100 SAVE "video"SCREEN$
 110 PRINT '"FERMA IL NASTRO": S
TOP
```

Si riporta il contenuto del video ottenuto con il comando COPY.



```
PROVA MEMORIZZAZIONE VIDEO
```

Segue un semplicissimo programma "CARVIDEO" che ripristina il contenuto del video leggendo il file di byte "video". Se prestate attenzione alla fase di caricamento del video vedrete le seguenti cose:

- il video viene costruito lavorando sulle 24 possibili linee a 8 per volta, prima da 0 a 7, poi da 8 a 15 e poi da 16 a 23,
- il fenomeno si osserva chiaramente, infatti il disegno nasce prima con la prima linea di pixel delle prime 8 linee di stampa, poi con la seconda linea e così via fino a completare le prime 8 linee di stampa,
- il disegno nasce nel colore di INK e viene colorato dopo, cioè gli ATTRIBUTI sono caricati per ultimo.

```
  20 PRINT "Monta nastro con fil
e video"'"Avvia il nastro e prem
i un tasto"
  25 PAUSE 0
  30 CLS : LOAD "video"SCREEN$
  40 STOP
```

Gli esempi riportati sono molto semplici, ma hanno solo lo scopo di far imparare queste tecniche che possono poi essere ampiamente sfruttate per costruire programmi anche molto complessi. Trattando il video si deve solo fare attenzione a non sporcarlo con cose non desiderate prima di memorizzarlo. Si ricorda che un disegno video può anche essere conservato altrove in memoria, e poi trasferito su video dal programma.

APPENDICE E

Codice Binario ed Esadecimale

Questa appendice spiega come il calcolatore conti col sistema binario.

La stragrande maggioranza delle lingue europee contano ripetendo più o meno regolarmente di 10 in 10 gli stessi prefissi e suffissi. In italiano, per esempio, anche se vi è qualche irregolarità all'inizio, contare diventa subito ripetitivo:

venti, ventuno, ventidue,..., ventinove
trenta, trentuno, trentadue,..., trentanove
quaranta, quarantuno, quarantadue,..., quarantanove

e così via, ancor più rigorosamente col sistema di numerazione Arabo, universalmente accettato. Se si è stabilizzato un sistema in base 10, è solo perchè abbiamo 10 dita. Il calcolatore, invece del sistema decimale (in base 10), usa un sistema binario (in base 2), che è comunque piuttosto scomodo per gli uomini, dato che consiste solo di 0 e 1. Data la estrema facilità di convertire numeri binari in numeri *esadecimali* e viceversa, tutto ciò che è binario viene generalmente indicato in esadecimale (abbreviato in *hex*), il sistema in base 16. Così come il sistema in base 10 ha 10 cifre, il sistema esadecimale ne ha 16; oltre alle dieci del decimale, usa A, B, C, D, E e F: dopo la F, esattamente come dopo il 9, nel sistema decimale, viene il 10, che avrà però il valore sedici invece che dieci.

Hex	*Italiano*
0	zero
1	uno
2	due
.	.
.	.
.	.
9	nove

regolarmente fino a nove, come in decimale, ma dopo continua con

A	dieci
B	undici
C	dodici
D	tredici
E	quattordici

F	quindici
10	sedici
11	diciassette
.	.
.	.
.	.
19	venticinque
1A	ventisei
1B	ventisette
.	.
.	.
.	.
1F	trentuno
20	trentadue
21	trentatre
.	.
.	.
.	.
9E	centocinquantotto
9F	centocinquantanove
A0	centosessanta
A1	centosessantuno
.	.
.	.
.	.
B4	centottanta
.	.
.	.
.	.
FE	duecentocinquantaquattro
FF	duecentocinquantacinque
100	duecentocinquantasei

È una buona regola, quando si usano numeri esadecimali, evidenziare che sono tali, scrivendo una "h" (per "hex") alla fine del numero. Per esempio, per centocinquantotto si scrive "9Eh", e si legge "nove E esadecimale".

Il sistema esadecimale è stato scelto per rappresentare il sistema binario, dato che, come detto, è facilmente convertibile, questo perchè 16 è una potenza di 2; ne consegue che ogni cifra esadecimale rappresenta esattamente quattro bit binari, e quindi segue che un byte è rappresentato da due cifre esadecimali, come si può apprezzare dalla seguente tabella:

Italiano	*Decimale*	*Esadecimale*	*Binario*
zero	0	0	0 o 0000
uno	1	1	1 o 0001

Italiano	*Decimale*	*Esadecimale*	*Binario*
due	2	2	10 o 0010
tre	3	3	11 o 0011
quattro	4	4	100 o 0100
cinque	5	5	101 o 0101
sei	6	6	110 o 0110
sette	7	7	111 o 0111
otto	8	8	1000
nove	9	9	1001
dieci	10	A	1010
undici	11	B	1011
dodici	12	C	1100
tredici	13	D	1101
quattordici	14	E	1110
quindici	15	F	1111
sedici	16	10	10000

Usando questa tabella potrete convertire qualsiasi numero esadecimale in un numero binario, o viceversa un numero binario in uno esadecimale, facendo attenzione a separarlo in gruppi di quattro bit, partendo da destra.

I numeri binari di 16 bits sono rappresentati comodamente con quattro cifre esadecimali; è il caso degli indirizzi, oppure delle parole, di lunghezza di due byte. Un byte è sempre formato da otto bits, ma le parole possono avere lunghezza diversa da calcolatore a calcolatore, e rappresentano sempre e comunque l'unità base della memoria del calcolatore stesso. Sullo ZX Spectrum l'unità di memoria è il byte stesso. La notazione **BIN** spiegata nel capitolo 23 permette di scrivere i numeri binari direttamente nella loro forma, **BIN 0** equivale a zero, **BIN 1** a uno e **BIN 10** a due, ecc. ovviamente, dato che dopo **BIN** potete usare sia 0 che 1, il numero che si otterrà sarà non-negativo ed intero, e comunque non più grande di 65535, ovvero non più lungo di 16 bits. Per far sì che **BIN** ritorni un numero negativo, usate, per esempio, **—BIN 11**, ma non **BIN —11**.

Convertire certe informazioni del calcolatore in binario ne facilita molto l'interpretazione, per esempio, il valore ritornato da **ATTR** presenta un numero binario a 8 bits, in cui il primo bit è 1 per lampeggiamento e 0 per carattere stabile, il secondo 1 per luminosità extra, 0 per normale, i seguenti tre formano il codice per il colore della carta in binario, gli ultimi tre per il colore dell'inchiostro, in binario.

Anche i codici dei colori sono scritti in binario; infatti ogni colore può essere scritto come combinazione dei tre colori primari; il primo rappresenta il verde, il secondo il rosso ed il terzo il blu. Il nero non è nessun colore, così tutti i suoi bits sono 0, mentre il bianco ha tutti i colori, così i suoi bit sono tutti 1.

I colori puri, il verde, il rosso, il blu hanno solo un bit a 1, e avranno quindi codici 100, 010 e 001, che in decimale valgono 4, 2 e 1. I colori non nominati sono il risultato dell'unione di due colori primari, e quindi i loro codici avranno due bit per ognuno.

APPENDICE F

Compatibilità ZX81

Possiamo considerare il Basic dello ZX 81 come un sottoinsieme del Basic dello Spectrum. Si hanno le seguenti differenze:

- mancano i comandi FAST e SLOW, lo Spectrum opera alla velocità FAST e trasmette i dati al video in SLOW senza far interferire le due operazioni tra loro, la gestione del video risulta indipendente;
- manca il comando SCROLL, e lo Spectrum fa lo scrolling automaticamente, ma con controllo dell'utente dopo la domanda "scroll?" che appare nella parte bassa del video;
- il comando UNPLOT manca e viene sostituito da PLOT OVER 1;
- i caratteri dello Spectrum sono ASCII;
- i programmi registrati su cassette dello ZX 81 non possono essere letti dallo Spectrum, ma i listati possono essere ribattuti e migliorati sfruttando le caratteristiche superiori del nuovo calcolatore;
- l'espansione da 16K RAM dello ZX 81 non può essere usata sullo Spectrum;
- la ZX Printer può essere attaccata allo Spectrum.

INDICE ANALITICO

In questo indice analitico si riporta il modo per ottenere le parole chiave del linguaggio Basic, cioè lo stato rilevabile dal cursore e il tasto SHIFT da usare. Di norma si riporta un solo riferimento nell'ambito di un capitolo, per cui si consiglia di riguardare tutto il capitolo indicato.

C

D

I

K

L

M

N

R

S

T

U

V

{	E, shifted F.	
|	E, shifted S.	
}	E, shifted G.	
~	E, shifted A.	
©	E, shifted P.	
<=	K, L o C, SYMBOL SHIFT Q.	260
>=	K, L o C, SYMBOL SHIFT E.	260
<>	K, L o C, SYMBOL SHIFT W.	760
◄	K, L o C, CAPS SHIFT 5.	59, 66, 143, 239, 251
►	K, L o C, CAPS SHIFT 8.	
▼	K, L o C, CAPS SHIFT 6.	
▲	K, L o C, CAPS SHIFT 7.	

L. 22.000 Cod. 337B ISBN 88-7056-135-6

Dopo il successo mondiale dello ZX81, ora la Sinclair ha presentato un nuovo microcomputer: lo ZX SPECTRUM.
Capirne il funzionamento e riuscire a sfruttarne appieno la potenzialità è quindi un modo intelligente per utilizzare da subito e al meglio questo piccolo calcolatore, piccolo però solo nelle dimensioni.
Nato dalla traduzione dei due manuali inglesi è costituito da ben 35 Capitoli.
Già il numero di questi fa capire l'impostazione del volume: massima segmentazione per ottenere massima elasticità di lettura (se si è esperti di programmazione alcuni capitoli possono essere trascurati), massima chiarezza espositiva e facile consultazione.
Dopo aver visto cosa sono i calcolatori, il lettore potrà già capire le differenze tra SPECTRUM e gli altri computer.
Imparerà poi a programmare in BASIC, (dai rudimenti del linguaggio alle funzioni avanzate del BASIC, alle possibilità grafiche e di animazione) e ad usare efficientemente il registratore per memorizzare e caricare i programmi (quelli presentati sono tutti provati e alcuni originali) e a sfruttare al meglio le stampe.

GRUPPO
EDITORIALE
JACKSON

di Rita Bonelli

Alla scoperta dello ZX Spectrum

80